1997-09

A relational database model and data migration plan for the student services department at the Marine Corps Institute

Slaughter, Aaron Tory

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A RELATIONAL DATABASE MODEL AND DATA MIGRATION PLAN FOR THE STUDENT SERVICES DEPARTMENT AT THE MARINE CORPS INSTITUTE

by

Aaron Tory Slaughter

September, 1997

Thesis Advisor: Magdi N. Kamel
Second Reader: Suresh Sridhar

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1997	Master's Thesis

4. TITLE AND SUBTITLE
A RELATIONAL DATABASE MODEL AND DATA MIGRATION PLAN FOR THE STUDENT SERVICES DEPARTMENT AT THE MARINE CORPS INSTITUTE

5. FUNDING NUMBERS

6. AUTHOR(S) Slaughter, Aaron Tory

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Marine Corps Institute
Arlington, VA 22222-0001

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(maximum 200 words)***

Today's business environment in the Department of Defense (DoD) demands that managers possess a clear understanding of the design, implementation, and maintenance of the databases used to store, organize, manipulate and return data.

In response to shortcomings identified in their current legacy information system, the Marine Corps Institute (MCI) initiated a project to migrate from a file processing database system to a relational database using a client/server system based on an open hardware and software architecture.

This research provides a relational data model and migration plan in response to MCI's request. It investigates data modeling and database design using the Integration Definition for Information Modeling (IDEF1X) methodology and the relational model. It also addresses the migration of data and databases from legacy to open systems. The application of the IDEF1X model, supported by CASE tools to facilitate data modeling and database maintenance, reveals strategies for dealing with the complex issues of database design, migration, and maintenance in DoD.

14. SUBJECT TERMS
Relational Databases, Data Modeling, IDEF1X, Open Systems, Client/Server

15. NUMBER OF PAGES 249

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI std. 239-18 298-102

Approved for public release; distribution is unlimited

A RELATIONAL DATABASE MODEL AND DATA MIGRATION PLAN FOR THE STUDENT SERVICES DEPARTMENT AT THE MARINE CORPS INSTITUTE

Aaron Tory Slaughter
Major, United States Marine Corps
B.A., Tulane University, 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
September 1997

ABSTRACT

Today's business environment in the Department of Defense (DoD) demands that managers possess a clear understanding of the design, implementation, and maintenance of the databases used to store, organize, manipulate and return data.

In response to shortcomings identified in their current legacy information system, the Marine Corps Institute (MCI) initiated a project to migrate from a file processing database system to a relational database using a client/server system based on an open hardware and software architecture.

This research provides a relational data model and migration plan in response to MCI's request. It investigates data modeling and database design using the Integration Definition for Information Modeling (IDEF1X) methodology and the relational model. It also addresses the migration of data and databases from legacy to open systems. The application of the IDEF1X model, supported by CASE tools to facilitate data modeling and database maintenance, reveals strategies for dealing with the complex issues of database design, migration, and maintenance in DoD.

TABLE OF CONTENTS

I. INTRODUCTION 1

A. BACKGROUND 1

B. OBJECTIVES 2

C. SCOPE AND METHODOLOGY 3

D. ORGANIZATION 4

II. BACKGROUND OF MCI, MCIAIS AND CONTEMPORARY SYSTEMS TECHNOLOGY 7

A. THE MARINE CORPS INSTITUTE (MCI) 7

B. THE MARINE CORPS INSTITUTE AUTOMATED INFORMATION SYSTEM (MCIAIS) 8
1. Current Data Environment 9
2. Current Data Environment Limitations 17
3. Current Processing Environment 19
4. Current Processing Environment Limitations 20
5. MCIAIS as a Legacy System 20

C. CLIENT/SERVER OPEN SYSTEM ENVIRONMENT 21
1. Client/Server Computing 22
2. Relational Databases and SQL 27
3. Open System Architecture 29
4. Benefits of Relational Databases in a Client/Server Open System Environment 30

III. METHODOLOGY AND TOOLS USED FOR DEVELOPING THE MCIAIS DATA ARCHITECTURE 33

A. DEVELOPMENT METHODOLOGY 33
1. List Candidate Data Entities 33
2. Define The Entities and Their Associated Attributes and Relationships 36
3. Develop the Data Model 36
4. Map Entities to Processes in the Process Model 37

B. IDEF1X MODEL 42

C. OVERVIEW OF ER*WIN* 48

1. Schema Generation 49
2. Schema Maintenance 49
3. Server Compatibility 50
4. Interface with the BP*win* Process Modeling CASE Tool 50
5. ER*win* ISSUES 51

D. OVERVIEW OF ORACLE 7 AND PL/SQL 53
1. Oracle 7 Architecture 53
2. PL/SQL Overview 57

E. AD HOC VERSUS FORMALLY DESIGNED DATABASES 60
1. Ad Hoc Designed Databases 60
2. Formally Designed Databases 60

IV. PROPOSED RELATIONAL DATA MODEL FOR MCIAIS 63

A. ENTERPRISE DATA MODEL FOR MCI 63
1. List Candidate Data Subjects 64
2. Define The Data Subjects and Their Associated Attributes and Relationships 68
3. Develop an Initial Conceptual Data Model 69
4. Development of Planning Matrices 69

B. MCIAIS DATA ARCHITECTURE FOR SSD AND MIS 70
1. Entities, Attributes, and Relationships 70
2. IDEF1X Model 74

V. PROTOTYPE RELATIONAL SCHEMA 79

A. GENERATED ORACLE SCHEMA 79
1. Enforcing Referential Integrity 81
2. Avoiding the Oracle Mutating Table Error 82
3. Specifying Data Definitions in ER*win* 82

B. SERVER SIDE TRIGGERS 83
1. Using Triggers to Enforce Referential Integrity 83
2. Using Triggers to Enforce Various Business Rules 85
3. Dangers Associated With Using Stored Procedures and Triggers 86

C. SCHEMA MAINTENANCE IN ER*WIN* 88

VI. DATA MIGRATION ISSUES 89

A. FORMAT AND LOCATION OF EXISTING DATA 89

B. DATA MIGRATION STRATEGY 90

C. DATA MIGRATION CONSTRAINTS 90

D. DATA MIGRATION IMPLEMENTATION 91

VII. CONCLUSIONS AND RECOMMENDATIONS 95

A. ACHIEVEMENT OF RESEARCH OBJECTIVES AND QUESTIONS 95
1. Primary Research Questions and Answers 96
2. Subsidiary Research Questions and Answers 96

B. SUMMARY OF DEVELOPMENT AND MAINTENANCE STRATEGIES 97

C. IMPLEMENTATION RECOMMENDATIONS 98

D. ANTICIPATED OBSTACLES 99

E. APPLICATION FOR OTHER SYSTEMS 99

F. FUTURE RESEARCH REQUIREMENTS 100

G. CONCLUSIONS 100

LIST OF REFERENCES 103

APPENDIX A. ACRONYMS AND TERMS 105

APPENDIX B. MCI ENTERPRISE LEVEL CANDIDATE DATA SUBJECTS 107

APPENDIX C. MCI ENTERPRISE LEVEL DATA SUBJECT DEFINITIONS 109

APPENDIX D. MCI ENTERPRISE LEVEL PRIMARY KEYS 111

APPENDIX E. MCI ENTERPRISE LEVEL CANDIDATE ATTRIBUTES 113

APPENDIX F. MCI ENTERPRISE LEVEL ATTRIBUTE DEFINITIONS 115

APPENDIX G. MCI ENTERPRISE LEVEL RELATIONSHIP DEFINITIONS 129

APPENDIX H. MCI ENTERPRISE LEVEL DATA MODEL 133

APPENDIX I. MCI ENTERPRISE LEVEL MATRICES 135

APPENDIX J. SSD AND MIS LEVEL CANDIDATE ENTITIES 139

APPENDIX K. SSD AND MIS LEVEL ENTITY DEFINITIONS 141

APPENDIX L. SSD AND MIS LEVEL ENTITY PRIMARY KEYS 145

APPENDIX M. SSD AND MIS LEVEL CANDIDATE ATTRIBUTES 155

APPENDIX N. SSD AND MIS LEVEL ATTRIBUTE DEFINITIONS 161

APPENDIX O. SSD AND MIS LEVEL RELATIONSHIP DEFINITIONS 185

APPENDIX P. SSD AND MIS LEVEL DATA MODEL 197

APPENDIX Q. STEPS REQUIRED TO AVOID THE ORACLE MUTATING TABLE ERROR WHEN USING ER*WIN* 199

APPENDIX R. SERVER SIDE TRIGGER PL/SQL SOURCE CODE 201

APPENDIX S. ATTRIBUTE AVAILABILITY REPORT 219

INITIAL DISTRIBUTION LIST 235

ACKNOWLEDGEMENTS

The author would like to acknowledge the financial support of the Marine Corps Institute in funding the travel and equipment purchased during the research phase of this thesis.

I. INTRODUCTION

The recent trend in migrating legacy, non-relational database systems to open, client/server relational databases has illustrated a clear need for better understanding of data modeling and database design methodologies, as well as Computer Aided Software Engineering (CASE) tools available to support data modeling and database schema generation. Additionally, the problem of data and database migration from legacy systems to client/server demands careful consideration and planning. Because technology is evolving so rapidly, a clear understanding of data and the associated database systems used to manage the data is critical to mission accomplishment. Today's business environment in the Department of Defense (DoD) demands that managers possess a clear understanding of the design, implementation, and maintenance of the databases used to store, organize, manipulate and return data.

This research investigates data modeling and database design using the Integration Definition for Information Modeling (IDEF1X) methodology and the relational model. It also addresses the migration of data and databases from legacy to open systems. The application of the IDEF1X model, supported by CASE tools to facilitate data modeling and database maintenance, reveals strategies for dealing with the complex issues of database design, migration, and maintenance in DoD. A case study of a mission-critical system at the Marine Corps Institute (MCI) illustrates the implementation of these strategies.

A. BACKGROUND

This research is the culmination of a year long project commissioned by MCI to develop the architecture and supporting migration plan to transition from a closed, non-

relational system to an open, client/server based relational database management system (DBMS). In response to shortcomings identified in their current information system, MCI initiated a project to migrate from a file processing database system to a relational database using a client/server system based on an open hardware and software architecture. MCI is also reviewing and redesigning business processes to better support its mission and current advances in training and education. Initial analysis of the current system and maintenance focused on both the data and processing environments.

B. OBJECTIVES

The primary questions to be answered by this research are:

- Can a data model be developed in IDEF1X to support the current and future needs of the student services department (SSD) at MCI?
- Can existing data in the Marine Corps Institute's Automated Information System (MCIAIS) be successfully migrated from the current legacy system to an open client/server system based on a relational database?

Subsidiary questions to be answered by this research are:

- How successful are available CASE tools in supporting data modeling using the IDEF1X methodology?
- How effectively do data modeling tools support other stages of the system development life cycle by interfacing with process modeling CASE tools?
- How effectively does the target DBMS support the need for a robust client/server based relational DBMS?

The process of answering these research questions reveals a common sense, logical strategy for accomplishing the design of a relational DBMS, the migration of existing data from a non-relational database to the new system, and the subsequent maintenance and evolution of the new system.

C. SCOPE AND METHODOLOGY

This research is part of a larger group project aimed at the development of a comprehensive architecture and migration plan in support of SSD at MCI. As such, this research was closely coordinated with the work of three other theses which addressed the following areas:

- Modeling and re-engineering business processes in SSD.
- Development of an open system technology architecture for MCIAIS.
- Development of a graphical user interface (GUI) application prototype for SSD.

This thesis focuses on data modeling and data migration aspects of the overall system. Data modeling is limited to the MCIAIS system of MCI and the necessary interfaces with the Logistics Automated Information System (LOGAIS) database and the Marine Corps Total Force (MCTF) database.

The intent of this research is to perform a detailed analysis of data requirements at MCI, review the existing non-relational database, and produce a new relational database to increase efficiency, reduce costs, and support future system evolution. Development of a conceptual data model, using the IDEF1X technique, and a relational database schema, is a primary goal of this research. Additional goals include the development of a data migration plan and comprehensive data dictionary in order to facilitate the migration effort.

In the course of accomplishing these goals, analysis of the effectiveness of the IDEF1X methodology and of a selected data modeling CASE tool is conducted. Also, the effectiveness of the target DBMS is reviewed. The accomplishment of these goals and

analysis reveals strategies relevant to all DoD organizations attempting to migrate from legacy file-processing systems to modern client/server relational systems.

The scope of this research and the accompanying data model and migration plan is limited to the development of entities, attributes, and relationships and the corresponding database schema necessary to support operations in the student services department at MCI. Further, this research is limited to the technical issues associated with implementing a relational database in to replace the existing legacy database. It does not examine organizational issues associated with system migration.

D. ORGANIZATION

This thesis is organized as follows: Chapter II presents an overview of MCIAIS as well as an examination of the trend toward open systems, with a goal of clarifying a working definition of the term "open system". This chapter also compares the current MCIAIS system with the benefits that could be enjoyed by employing a relational database in a client/server environment.

Chapter III describes the methodology advocated for the development of a target relational database. It also presents an overview of the IDEF1X modeling methodology, as well as a review of the CASE tool selected for this research. The chapter concludes with an overview of the target DBMS for this project and the associated procedural computer language.

Chapter IV presents the proposed data model for MCIAIS. It describes the data requirements, as well as the developed entities, attributes, and relationships. Finally, it relates these elements in an overarching IDEF1X model.

Chapter V discusses the prototype relational schema and associated triggers. This includes a discussion of translating the conceptual IDEF1X model to a logical relational design schema and the benefits of utilizing server side triggers to enforce business rules. It concludes with a discussion of the maintenance requirements associated with the target system.

Chapter VI addresses the migration issues from the current legacy data to the future relational database. It focuses on the areas of migration strategy, constraints, and system implementation.

Chapter VII presents conclusions and recommendations. It summarizes the various practical strategies for developing and maintaining relational databases. It presents recommendations for implementation of relational systems, and summarizes the technical obstacles to implementation. Additionally, applicability of this research to other systems at MCI (e.g., LOGAIS) is briefly discussed. Finally, future research requirements and opportunities for further research are suggested.

II. BACKGROUND OF MCI, MCIAIS AND CONTEMPORARY SYSTEMS TECHNOLOGY

The history of MCIAIS is that of a firmly entrenched legacy system with many shortcomings. These are in stark contrast to the growth potential, scalability and flexibility normally associated with client/server open systems. In order to develop a target architecture based on open systems for MCIAIS, it is necessary to review the history of both MCI and MCIAIS, and discuss the issues of migrating to a client/server open system. In particular, a comparison of the current MCIAIS system with a proposed client/server open system reveals the benefits that could be enjoyed by employing a relational database in an open client/server environment.

A. THE MARINE CORPS INSTITUTE (MCI)

The Marine Corps Institute was established in 1920 to "develop, publish, distribute, and administer distance training and education materials to enhance, support, or develop required skills and knowledge of Marines and to satisfy other training and education requirements as identified by the Commanding General, Marine Corps Combat Development Command."

To accomplish its mission, MCI is organized into six functional departments: the education and operations department (OPS), the professional military education department (PMED), the occupational specialty department (OSD), the student services department (SSD), the management of information systems department (MIS), and the logistics department (LOG).

The student services department's mission is to support the enrollment, grading and management of the Marine Corps distance education and training programs. In support of its mission, the student services department employs an automated information

system (AIS) to automate the actions required to support a student in the MCI correspondence program, maintain student records, and produce necessary management reports.

B. THE MARINE CORPS INSTITUTE AUTOMATED INFORMATION SYSTEM (MCIAIS)

The main information system used by MCI, known as the Marine Corps Institute Automated Information System (MCIAIS), is a legacy system developed in the late 1970's. It runs on a Hewlett Packard (HP) 3000 minicomputer running the MPE/iX operating system. MCIAIS is written in the HP proprietary language "Transact" and accesses an HP proprietary TurboIMAGE hierarchical database.

As typical of many legacy systems, MCIAIS suffers from many shortcomings:

1. It has over 110 "spaghetti coded" programs that are difficult to maintain, modify, and evolve.

2. It does not have underlying data or process models.

3. Programs have poor functionality, including:
   - Poor checks and balances;
   - No statistical analysis capability;
   - Limited ad hoc query capability.

4. It utilizes a "closed" non-relational database.

5. It does not support graphical user interfaces.

6. It does not follow DoD or United States Marine Corps (USMC) standards.

In response to these shortcomings, MCI initiated a project to redesign and rewrite MCIAIS using a client/server system based on an open hardware and software architecture. MCI is also reviewing and redesigning business processes to better support its mission and current advances in training and education. Initial analysis of the current

system and maintenance focused on both the data and processing environments. Limitations of both environments are hampering MCI's ability to perform its mission.

1. Current Data Environment

The current MCIAIS data environment consists of seven HP TurboIMAGE hierarchical databases. Additionally, MCIAIS interfaces with the LOGAIS database, which resides on the same minicomputer. Each of these databases consist of various records, which are flat files of assorted lengths. These databases are shown in Figure 2-1.

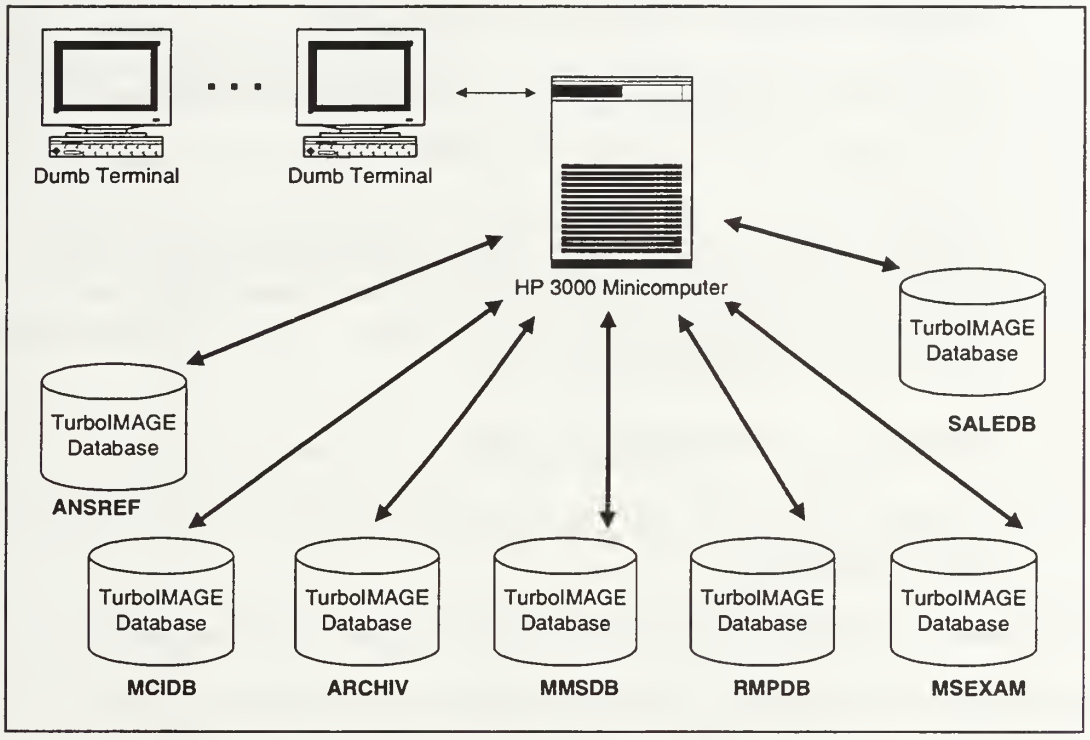

Figure 2-1. MCIAIS Databases

The current structure of these databases is the result of a redesign effort that began in 1990, when the requirement was established to redesign the MCI Automated Support System (MCIASS). MCIASS was redesignated as MCIAIS, and the databases and automated processes of MCI were redesigned based on a series of TurboIMAGE

databases. The TurboIMAGE databases were intended to allow for the automation of all actions required to support students, maintain student records, and produce necessary management reports. Databases maintained by MCIAIS include:

- ANSREF. Contains all answers and references for every course exam and lesson.
- MCIDB. Contains all student course and information records.
- ARCHIV. Contains thirteen month (or older) inactive student records.
- MMSDB. Contains information on the records of active duty Marines. Used to construct and update student records. Originally extracted from the Marine Corps Manpower database. Now extracted from the MCTF database 3 times a month during the "MCTF Download" batch process.
- RMPDB. The sister database to MMSDB. Contains information on the records of reserve Marines. Used to construct and update student records. Originally populated from the Reserve's Manpower Management System Database. Now extracted from the MCTF database 3 times a month during the "MCTF Download" batch process.
- MSEXAM. Contains information on the on hand status of exam stock.
- SALEDB. Stores the raw data for the Statistical Analysis of Lessons and Exams (SALE) report.

These databases are described briefly in the following sections to give the reader a clearer understanding of the current data environment and the subsequent efforts of developing a comprehensive data model that incorporates the data from these databases.

a. ANSREF Database

ANSREF-DETL, the primary data set of ANSREF, contains exam grading information and the references for each exam question. As MCI no longer grades lessons,

only the exam information is relevant. Table 2-1 depicts the layout of the ANSREF-DETL database.

FIELD NAME	LENGTH	USED	DESCRIPTION
* COURSE	6	3-5	Course number and version
*LES-EXAM	2	1	Lesson number or Exam letter
*ANSWERS	160	1-160	Array of answer codes
REFERENCES	3200	0-3200	Array of references
	3368 = record length for ANSWREF-DETL		
	* denotes the field must always have a value		

Table 2-1. Layout for ANSREF-DETL Record

Each exam consists of 1 to 160 multiple choice questions. Answer codes indicate possible correct answers. These answer code definitions are shown in Table 2-2.

ANSWER CODE	DESCRIPTION
A	Answer to be graded as A
B	Answer to be graded as B
C	Answer to be graded as C
D	Answer to be graded as D
E	Answer to be graded as E
F	Answer to be graded as A or B
G	Answer to be graded as A or C
H	Answer to be graded as A or D
I	Answer to be graded as A or E
J	Answer to be graded as B or C
K	Answer to be graded as B or D
L	Answer to be graded as B or E
M	Answer to be graded as C or D
N	Answer to be graded as C or E
O	Answer to be graded as D or E
X	Any answer, including blank, is correct

Table 2-2. Answer Code Definitions

It is important to note that answer code definitions are not stored in the database, but are embedded in the programs that access these databases. Additionally, possible

combinations of acceptable answers exist which are not defined. For example, if possible correct answers for a specific question are defined as A, B, or C, no code exists which matches this combination of acceptable answers.

b. MCIDB Database

MCIDB is the primary database in MCIAIS. It contains all records about a student's progress through an MCI course. This database also contains all records relating to course information. The following primary data sets are contained in MCIDB:

- MCOURSE. This data set contains information for each individual course, including some outdated information which MCI business rules no longer require (e.g., course weight). Every MCI course has a corresponding record in this data set. Table 2-3 depicts the layout of an MCOURSE record.

- MRUC. This data set contains the address for every Reporting Unit Code (RUC) in the Marine Corps. It also contains outdated information no longer required by MCI business rules, such as information regarding whether a unit receives a unit audit report (UAR) and the last date the UAR was returned. Table 2-4 depicts the layout of an MRUC record.

- STUD-INFO-DETL. This data set contains the student's personal information. Each student has a corresponding record in this data set. Table 2-5 depicts the layout of a STUD-INFO-DETL record.

- STUD-CRS-DETL-A. This data set contains all active student records. There is one record for student-course enrollment. This data set is used to track the student's progress in a course. When the student completes, fails, or is disenrolled from a course, this record is removed and placed in the history data set (STUD-CRS-DETL-H). Table 2-6 depicts the layout of a STUD-CRS-DETL-A record.

- STUD-CRS-DETL-H. This data set contains the same fields as STUD-CRS-DETL-A with the addition of the RUC field. Students who have completed, failed or where disenrolled from a course are placed in this data set. After thirteen months the student is removed and placed in the archive (ARCHIV) database.

FIELD NAME	LENGTH	USED	DESCRIPTION
* COURSE	6	3-5	Course number and version
*TITLE	60	60	Title of course
*CRS-ABBR	20	20	Abbreviation used for MMS updates
*WEIGHT	4	4	Weight in pounds (P) and ounces (O). PPOO format
DATE-CRS-OPEN	6	0,6	Date opened (YYMMDD)
DATE-CRS-CLOSE	6	0,6	Date Closed (YYMMDD)
*NUM-LESSONS	2	1	Number of lessons
*CATEGORY	2	1	Category code
*STATUS	2	1	Status code
*EX1FORM	2	1	Primary exam form (A-Z)
*EX2FORM	2	1	Alternate exam form (A-Z)
DATE-EX1-OPEN	6	0,6	Date primary exam opened
DATE-EX2-OPEN	6	0,6	Date alternate exam opened
*RESERVE-CREDITS	2	2	Reserve credits for USMC
*VCOURSE	4	4	Course number without version number.
*CONT-FLAG	2	1	For PMED courses. Not used.
*QUANTITY	8	8	Quantity currently in stock
ISSUED-TODAY	4	4	Quantity issued today
	144 = record length for MCOURSE		
	* denotes the field must always have a value		

Table 2-3. Layout for MCOURSE Record

FIELD NAME	LENGTH	USED	DESCRIPTION
* RUC	6	5	Reporting unit code
MCC	4	0,3	Major command code
*ADDR1	40	1-40	First address line
*ADDR2	40	1-40	Second address line
*CITY	20	1-20	City
*STATE	2	2	State abbreviation
*ZIPCODE	10	10	Zip code in #####-####
*UAR-RECEIVER	2	1	Y or N, does unit receive UAR
UAR-RETURN-DATE	6	0,6	date last UAR returned
RUC-FLAG	2	0	Not implemented
	132 = record length for MRUC		
	* denotes the field must always have a value		

Table 2-4. Layout for MRUC Record

FIELD NAME	LENGTH	USED	DESCRIPTION
* SSN	10	9	Student's social security no.
LASTNAME	20	1-20	Student's last name
*INITIALS	2	1-2	Student's first and middle initials
RANK	6	0-6	Student's rank
GRADE	2	0,2	Student's grade
*RUC	6	5	Student's RUC
MOS	4	0,4	Student's primary MOS
*COMPONENT	2	1	Component Code
OLD-RUC	6	0,5	Student's previous RUC
DATE-RUC-CHG	6	0,6	Date OLD-RUC modified

64 = record length for MRUC
* denotes the field must always have a value

Table 2-5. Layout for STUD-INFO-DETL Record

FIELD NAME	LENGTH	USED	DESCRIPTION
* SSN	10	9	Student's social security no.
*COURSE	6	3-5	Course number and version
*ENROLL-DATE	6	6	Enrollment date in YYMMDD
REENROLL-DATE	6	0,6	Re-enrollment date
*DEADLINE	6	6	Course completion date
EXTENDED	2	0,1	Y or N student extension flag
MOTOVATION	2	0,1	code for last motivation notice
LESSONS	10	0-9	Pass or fail lesson. Array of 9.
EX1FORM	2	0,1	Primary exam form A-Z
EX1SENT	6	0,6	Date primary exam mailed
EX1SCORE	4	0-3	Primary exam score percent
EX2FORM	2	0,1	Secondary exam form A-Z
EX2SENT	6	0,6	Date secondary exam mailed
EX2SCORE	4	0-3	secondary exam score percent
SUBCOURSES	220	0,220	PMED courses in array of 20
*LASTTRAN	8	7	Last transaction code and date
TRANARRAY	98	0-98	Last 14 transactions and dates
*PME-FLAG	2	1	Y or N. Identifies PMED courses
*MM-USER-DATE	14	14	Last user to modify record and date occurred, or the program that created the record

414 = record length for STUD-CRS-DETL-A
* denotes the field must always have a value

Table 2-6. Layout for STUD-CRS-DETL-A Record

Similar to the ANSWERS field in the ANSREF-DETL record, many coded fields exist in the MCIDB for which codes are not defined in the database. Examples of coded fields with no definitions present in the database include CATEGORY and STATUS from MCOURSE, RANK, GRADE, and COMPONENT from STUD-INFO-DETL, and MOTOVATION and LASTTRAN from STUD-CRS-DETL-A. Similarly, states are entered as two letter abbreviations, with no record of corresponding state names stored in the database. Additionally, reporting unit codes are stored without corresponding unit names.

c. ARCHIV Database

ARCHIV is the archival database for student course records on STUD-CRS-DETL-H (from MCIDB) that have been inactive for 13 months. Currently, there is no specified length of time that records will remain on ARCHIV. Once a student has been disenrolled for thirteen months (i.e., a record has existed in STD-CRS-DETL-H for thirteen months), portions of the record are transferred to the ARCHIV database and the original record is deleted. This is done for two reasons: Conservation of disk space and to keep the size of MCIDB manageable. The ARCHIV database contains only the fields necessary to create transcripts and verify student course completion.

d. MMSD and RMPDB Databases

These two databases were first implemented in 1987 in an attempt to improve the accuracy of the student data in the database. Of particular concern was the fact that when student's changed units, they failed to notify MCI of their new mailing address. As a remedy, MCI elected to only mail Marine student's material to addresses downloaded from Marine Corps manpower databases. Currently, all manpower

information (both active and reserve) is maintained in the MCTF database. MCI extracts this data periodically and loads the MMSD and RMPDB databases.

These databases have a lifetime equal to the time between downloads. Each download is loaded to a temporary database. Once the data has been validated, all existing data in MMSD and RMPDB is replaced.

e. MSEXAM database

The MSEXAM database is a logistics database which maintains information on exam version quantities in stock and the status of various exam versions. Like the databases discussed previously, this database contains records based on various codes which are not clearly defined in the database. An example of such a code is the exam status field.

f. SALEDB Database

This database stores raw data for statistical reports. The intent of this database is to store data necessary to formulate the SALES report. As such, it is designed to store data in three distinct records: A master record, a detail record, and an index record.

The master record contains a single field which stores a course number and version. The detail record contains the course number, the exam number, the number of exams graded per month, the number of failures per month, and the frequency of failures. The index record stores the course number , the exam number, the raw score for the exam instance, and the answer code array containing answer codes to all questions for the exam instance.

2. Current Data Environment Limitations

MCIAIS is based on a file-processing system. Like all file-processing systems, MCIAIS suffers from several limitations:

- Data is separated and isolated.
- data is often duplicated.
- Application programs are dependent on file formats.
- File types are often incompatible.
- Using models to represent data as the user views it is difficult. [Ref. 1]

a. Shortcomings of File Processing Systems as They Apply to MCI

At MCI, data is separated and isolated, which necessitates the combination of data from assorted files to present a new, useful view of the data to the user. Programmers must select which parts of the various files are related, and create the new view by correctly processing various parts of separated, isolated files. This is a tedious, burdensome, and often impossible task.

Various data elements within MCIAIS are duplicated in several files. This duplication wastes storage space. However, the most critical shortcoming of data duplication is the impact on data integrity. Data inconsistencies result when duplicate data elements are modified in some locations, and unmodified in other files. The question soon arises as to which version of the duplicate data is accurate.

MCIAIS has spawned applications which are dependent on the specific proprietary file format associated with its non-relational databases. When changes are made in specific file formats, all associated applications must be changed as well.

More troublesome is the fact that sharing data is especially difficult. File formats between MCIAIS and various other systems such as MCTF and LOGAIS are incompatible, and require complex and time consuming reformatting in order to share data between various applications.

Retrieval of data is complicated by the lack of a structured query language (SQL) which standardizes the procedures used to retrieve data. As a result, procedural language programming support is required for all data retrieval.

Finally, it is difficult to represent data structures to users of MCIAIS. Because the non-relational databases are poorly structured, they are difficult to illustrate graphically. There are no commonly understood modeling techniques used to represent file-processing systems. Relationships among records are poorly documented and not easily understood.

b. MCIAIS as a Legacy System

MCIAIS can be considered a *legacy system.* A legacy system is one that "significantly resists modification and evolution to meet new and constantly changing business requirements [Ref. 2]." Legacy systems are expensive, inflexible, and consume enormous maintenance resources. It is estimated that operating and maintaining legacy systems can consume 80 to 95 percent of an IS budget. [Ref. 2]

At MCI data is duplicated, and data integrity must be enforced with programs. Keeping track of the various data sets and maintaining the integrity of duplicate records requires a large programming maintenance effort. None of the data conforms to USMC or DoD standards, and is incompatible with other databases of

dissimilar file types. Finally, data relationships are not clear, and cannot be modeled graphically.

3. Current Processing Environment

The declared purpose of the original MCIAIS TurboIMAGE application is the automation of all actions required to support a student in the MCI correspondence program, maintain student records, and produce necessary management reports. Every effort was made by the original MCIAIS design committee to eliminate the need for user error checking. All edit logic, decision logic and update logic related to the databases was defined explicitly in programs. Transactions containing defined edit errors are brought to the users attention for correction. Erroneous transactions that are entered in daily or monthly batch processes are rejected and appear on the appropriate error report. Thus, all business rules associated with the databases are enforced via Transact programs.

MCIAIS is composed of 110 largely undocumented application programs written in 1978 by MCI Marine programmers. MCIAIS has not been thoroughly re-engineered or redesigned in the nineteen years since it was first written. The redesign effort which began in 1990 has consisted of patching original programs to correct deficiencies and add new functionality. Additionally, MCI does not have adequate internal resources to properly maintain the MCIAIS system. This situation has contributed to the problems associated with patching existing programs and poor program documentation. As a result, there are numerous inconsistencies in programming logic which have produced inaccurate and unreliable information. Consequently, MCI frequently does not depend on the data provided by the MCIAIS system when making major management decisions.

4. Current Processing Environment Limitations

The current processing environment at MCI is a paper based, manually intensive, partially automated environment that is not capable of responding quickly to students' changing training, educational, and administrative needs. Further, the current system consists of non-USMC standard hardware and software, which causes incompatibility problems and undesirable training expenses with new personnel.

Because of sometimes inaccurate computer logic supporting outdated business practices, MCI admits that its customer service is greatly degraded [Ref. 4]. This situation causes considerable and serious impact across the Marine Corps. Current regulations closely tie retention and promotion in the Marine Corps to the successful and timely completion of MCI courses and programs.

5. MCIAIS as a Legacy System

As a typical legacy system, MCIAIS provides for poor separation of data, process and interface components of the overall system. Specifically, MCIAIS fails to separate the presentation, application, and data logic. Data, applications which manipulate that data, and logic which presents that data to the user are intertwined and poorly defined. There is no logical or physical separation of application programs, presentation programs, and database programs and files. This has contributed to the poor functionality of MCIAIS and is typical of legacy systems which are not based on the client/server model. Figure 2-2 illustrates the shortcomings of the present system: Presentation, application, and data logic are not separated in each of the applications.

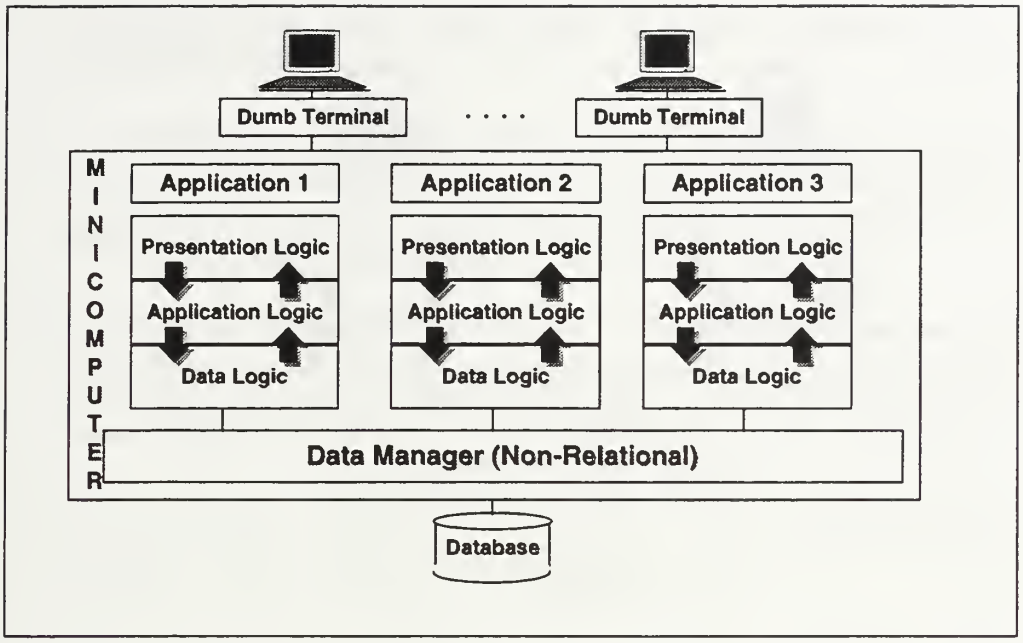

Figure 2-2. Existing Architecture

C. CLIENT/SERVER OPEN SYSTEM ENVIRONMENT

Many of the obstacles presented by traditional file-processing systems such as MCIAIS, which fail to separate presentation, application, and data logic, can be overcome by migrating to client/server based open systems. The multi-layer architecture of client/server open systems provides a logical separation of presentation, application, and data logic.

Transforming legacy systems into a contemporary, client/server, open architecture provides several advantages to users. These include streamlining operations with simultaneous multiple client data access, while providing local processing capabilities from individual workstations. The use of client workstations provides the benefit of GUIs for understandable presentation of applications and data. Additionally, users have the advantage of storing data in a relational database. Figure 2-3 illustrates how these systems are able to clearly separate the presentation, application, and data layers.

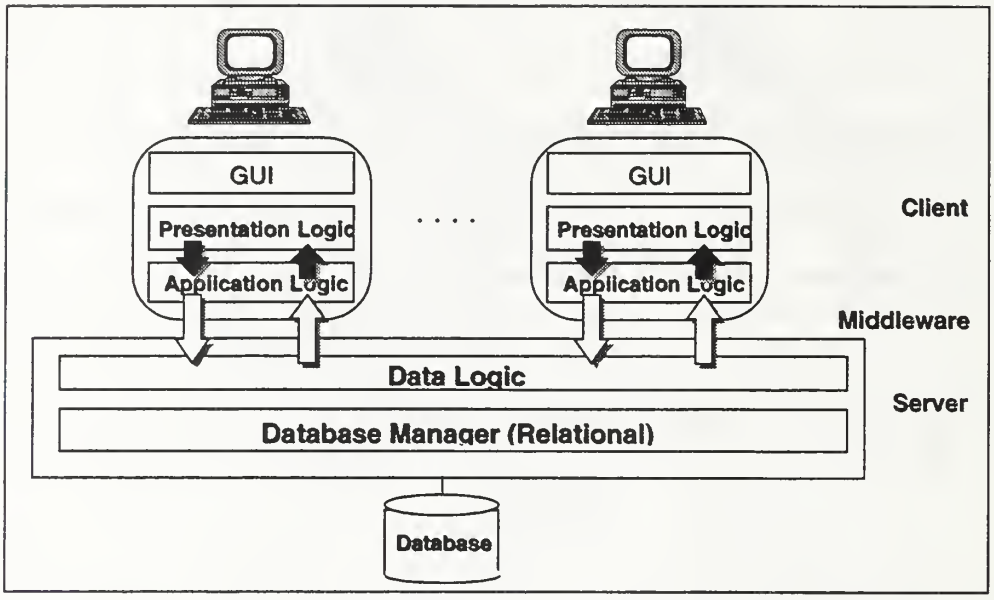

Figure 2-3. Target Client/Server Open Architecture

1. Client/Server Computing

Client/server computing is a general term that means different things to different people. In most cases, client/server computing implies distributing computing on clients, which normally possess their own processors and some degree of memory and secondary storage, and servers, which store various data and applications and also have processors. Clients and servers are considered *logically separate entities.*

a. Distinguishing Characteristics of Client/Server

Client/server can be distinguished from other forms of distributed computing by the following characteristics:

- Service: By definition, client/server implies a relationship between processes which are logically separated. Services are provided by servers and consumed by clients. This clean separation between functions based on service distinguishes client/server from other forms of distributed computing.

- Shared resources: Server resources can be accessed simultaneously by multiple clients.

- Asymmetrical protocols: This means that servers are passive in terms of service requests. Clients always initiate the request for service from the server. This distinction is becoming less clear with the advent of push technology applications on internets and intranets.

- Transparency of location: The location of the server should be transparent to the client. Servers can reside on the same machine as the client, or on the other side of the world. The software masks the location of the server from the client.

- Mix and match: This is another term for open systems. Open systems generally consist of non-proprietary components which can be mixed and matched. Open systems are further described below.

- Message-based exchanges: Clients and servers interact by passing messages back and forth. Service requests and replies are based on message traffic. In this sense, clients and servers are considered to be "loosely coupled".

- Encapsulation of services: Servers specialize in one or more types of services. These services are logically separated. For example, services can provide database service, file-processing service, electronic mail service, or any number of other services.

- Scalability: This is a key feature of client/server systems. Systems may be scaled vertically or horizontally. Horizontal scaling implies adding or removing clients. Vertical scaling means migrating to larger or faster servers, as the business enterprise expands.

- Integrity: This is another key advantage to client/server, especially when migrating from a legacy, non-client/server, file-processing database system. Integrity in terms of client/server database services implies that server code and data is logically separated from client applications. The middleware is the glue, in the form of software, which ties the two together. This results in cheaper maintenance and improved data security and integrity. [Ref. 5]

These client/server characteristics provide a framework for understanding the advantages that migrating to a client/server database computing environment hold. Clearly, these advantages are maximized when the database is relational in nature. In that

respect, presentation logic, application logic, and data logic can be cleanly separated between clients, middleware, and servers.

b. Fat versus Thin Clients

Client/server computing can be distinguished by how application logic is split between the clients and the server. As depicted in Figure 2-4, fat servers place most application functions on the server, while fat clients do the opposite.

Fat clients are the traditional form of client/server, including database servers [Ref. 5]. In this arrangement, clients typically know how data is organized and stored on the server. Data is retrieved from the server by the client and then manipulated by applications on the client side.

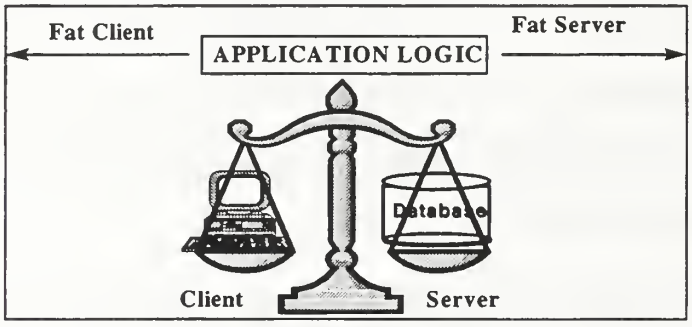

Figure 2-4. Fat Client vs. Fat Server, After Ref. [5]

Despite the popularity of the fat client approach, fat servers provide great advantages, especially with database servers. Fat server applications are easier to manage and deploy on networks because application code resides on the server. Changes in business rules require programming changes in one central location, rather than modification of software on every client in the network. The client in this model interacts with the server through GUI based remote procedure calls. [Ref. 5]

c. GUI clients

GUI clients are typical in modern client/server environments, and are a distinguishing characteristic of most client/server systems. GUI clients provide requests to the server by means of a human operator interacting with GUI presentation logic. These interfaces are the right choice for mainstream business applications which consist of high volumes of repetitive tasks. The simplest GUI applications are graphical renderings of the dialog screens that formerly ran on the legacy system's dumb terminals. GUI clients may also be based on the object action model, where users select an object from a set and then chose an action to perform on that object. [Ref. 5]

d. Servers

Servers in a client/server environment can fulfill many roles. These include file server, electronic mail server, transaction server, and database server. A typical database server solution involves the use of GUI clients to access relational database servers. With relational database servers, requests for information are passed as messages to the database server. Responses to the SQL requests are returned to the client over the network. Requests can be in the form of complete SQL statements or the client can pass procedure calls that trigger SQL statements stored on the server. The former technique is useful for processing ad hoc data queries, while the latter conserves network capacity by reducing the size and duration of message traffic between a client and the server. An important characteristic of this process is that the code which processes the SQL request and the data both reside on the server. The server uses its processing power to find requested data, instead of passing all records back to the client. Therefore, processing on the client is conserved. This is depicted in Figure 2-5. [Ref. 5]

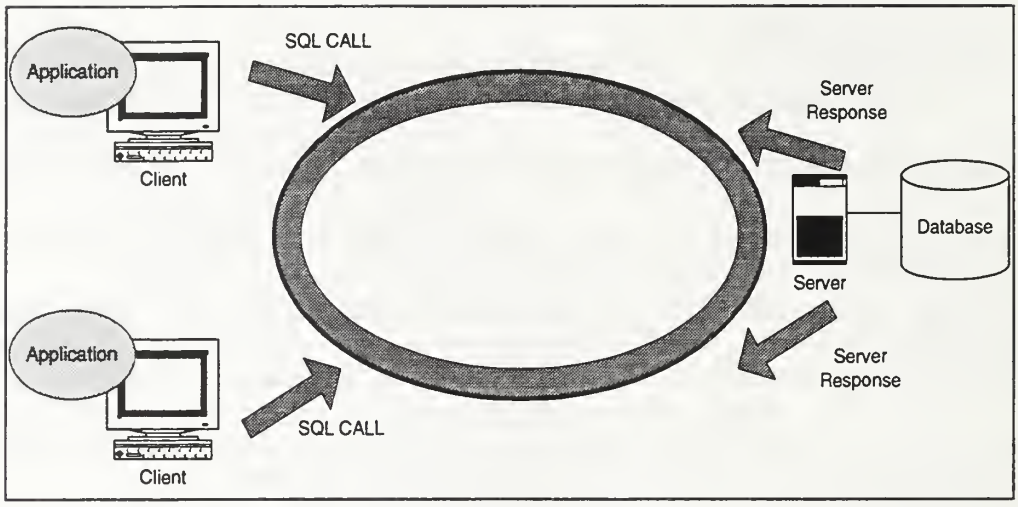

Figure 2-5. Client/Server Database Server, After Ref. [5]

e. Middleware

Middleware is a further characteristic of client/server computing environments. This term describes all the distributed software that is required to support the interaction between clients and servers. Middleware begins with the application programming interface (API) which allows clients to invoke services, continues with the transmission of the service request over the network, and concludes with the resulting transmitted response. It does not include the software which provides the actual service, nor does it include the user interface. [Ref. 5]

General middleware is not service specific, and is required in some fashion on all client/server systems. It includes communication stacks, directories of network nodes, services for providing validation and authentication of requests and responses, remote procedure calls, and prioritization queuing services. In addition to general middleware, some middleware is required based on the type of service provided by the server. This is known as service specific middleware. For database server environments, it includes the software required to access specific databases. Common examples include

Open Database connectivity (ODBC), Integrated Database Application Programming Interface (IDAPI), and Oracle Glue.

2. Relational Databases and SQL

One of the most important recent trends in database servers has been the emergence of SQL as the *de facto* standard for data manipulation, definition, and control [Ref. 6]. SQL has been standardized by the International Standards Organization (ISO), and is a powerful, non-procedural language which is easily understood. SQL is designed to interact with databases which conform to the relational database model.

a. The Relational Database Model

The relational database model is attributed to E. F. Codd, who developed it while working at IBM's San Jose, California research lab in the early 1970s. Codd's definitive paper on the relational model was published in June 1970 [Ref. 7]. This paper first introduced the relational approach, and proposed the notions of relational data structures based on a mathematical model.

A relational database is simply a database that is perceived to be a collection of tables, where a table is an unordered set of rows. The word relation is a mathematical term for such a table [Ref. 8]. Mathematically, a relation can be defined as:

> Given sets S1, S2, . . . , S*n* (not necessarily distinct), R is a relation on these *n* sets if it is a set of *n*-tuples, the firs component of which is drawn for S1, the second component from S2, and so on. [Ref. 9]

Each of the sets S1, S2, . . . , S*n* on which one or more relations are defined is known as a domain. Therefore, the domain is the set of all values for a particular attribute [Ref. 6].

Two special properties characterize mathematical relations. First, all elements of a relation are tuples, and all are of the same type. Second, a relation is an unordered set. A relation R in Codd's relational model is very similar to a mathematical relation. When modeled as a table, a relation R in Codd's model has the following special properties:

- Each row represents a tuple of R;
- the ordering of the rows is immaterial;
- All rows are distinct from one another in content; [Ref. 9]

Thus, rows that entirely duplicate each other are not permitted in relational databases. This characteristic is enforced by identifying one or more columns as the primary key to the relation. Primary key values may not be duplicated within a relation. This disciplined approach is necessary in order to maintain the integrity of the data. Data integrity is a primary concern in relational databases, because these databases are often shared by many users. [Ref. 9]

b. Structured Query Language

SQL is a non-procedural data access language. A non-procedural language is used to specify what data is needed, but not how to perform the task of retrieving the data. The decision on how to execute the SQL request is made by the DBMS. Particular implementations of SQL may differ in minor ways from the ISO and American National Standards Institute (ANSI) standard, yet all follow the same basic constructs.

SQL commands can be used directly to interact with the database, or they can be embedded in application programs which are stored on either the client or the server [Ref. 1]. All SQL data queries accept one or more relations as input and produce a

single relation as output. Thus, the result of every SQL query is a relation; even if the result is a single number, that number is considered to be a relation with a single row and column [Ref. 1].

3. Open System Architecture

One method of designing client/server systems is to adopt an open system architecture. Like client/server, the phrase "open system" means different things to different people. Generally, open systems refer to systems that consist of components which are non-proprietary in nature. These components adhere to protocols which have been standardized formally by recognized national or international organizations. In some instances, open systems refer to protocols which, though proprietary in nature, have emerged as the industry standard, and are supported by numerous vendors' applications.

Analysis of the components of a system, including applications, hardware, operating systems, databases, and data can reveal indications as to the degree of openness of the overall system. [Ref. 3]

Openness of applications is measured in terms of application programming language, availability of source code, ability to modify code, availability of APIs, and degree of application documentation. Hardware can be used to measure openness in terms of operating systems supported, proprietary of non-proprietary nature of components, and whether the hardware components are interchangeable.

Network hardware on open systems should be able to support both vertical and horizontal growth, support multiple platforms running multiple operating systems, and support multiple network protocols. Openness of operating systems (client, network, and server) can be measured in terms of the type and number of platforms that can run the OS.

Database openness can be measured in terms of what APIs are available to access the DBMS, as well as how compliant the specific version of SQL employed by the DBMS is with the ANSI or ISO standard. It can further be measured in terms of interoperability with other DBMS engines.

Data is a critical component of open systems. Openness can be measured by how easily the data can be accessed by applications and users across the network. Closed systems are those which are proprietary in nature and do not allow manipulation and conversion of data to usable formats. Open systems frequently store data which is compliant in type and form to a standardized, enterprise-wide, data dictionary.

4. Benefits of Relational Databases in a Client/Server Open System Environment

Client/server computing coupled with a relational database yields numerous user benefits. These system benefits include additional functionality, ease of future modification, migration or expansion, and ease of data maintenance.

a. Additional Functionality

In a client/server environment centered on a relational database server, applications are used to request data and data services, such as sorting and filtering, from the relational database server. The most noticeable advantage from the user's perspective is the incorporation of a GUI front end on the client, which greatly increases the user's ability to intuitively interact with the system. This can greatly the reduce training time required to develop proficient users, as well as increase the users' productivity levels.

The database server responds to client requests by providing secured access to shared data. This is ability to securely share data is a major benefit of

client/server computing. Also, the server's ability to filter query result sets can result in considerable data communication savings [Ref. 5]. Because server resources can be accessed simultaneously by multiple clients, additional processing savings can be realized.

The ability to enforce business rules on the server, with the use of stored procedures, functions, triggers, and SQL based rules, allows applications to be distributed on thin clients. This approach minimizes the processing required on the client by centralizing data processing requirements on the server. Centralization of business rule enforcement mechanisms can result in additional cost savings, and reduces the chances of users modifying business rule enforcement mechanisms and inserting non-compliant data without authorization.

b. Ease of Future Modification and Expansion

Enforcing business rules on the server eases labor requirements when modifications to business rules are required. Applications on distributed clients require minimal modification when business rules are changed if the code which enforces those rules is stored and executed on the server. In many cases, business rules can be modified without requiring modification and redistribution of applications. However, it is important to note that applications will require modification if new business rules require the use of data elements not previously identified to the client applications. Therefore, it is important that database design be comprehensive, and include all potential data elements prior to system deployment.

Expansion of the system in a client/server environment is readily achieved as well. Horizontal and vertical scaling are easily accomplished. When the system is

based on an open architecture, clients conforming to the open system specifications can be seamlessly integrated. Unneeded clients can be removed from the network at anytime without disrupting service. As data processing requirements are increased, migration to larger, faster, more capable servers can be smoothly accomplished.

c. Ease of Maintenance

Data maintenance requirements are reduced in a client/server environment. Because data is centrally stored and managed, maintenance is required in only one location. Integrity is a key advantage to client/server when migrating from a file-processing database system. Because server code and server data is logically separated from client applications, data security and integrity is strengthened. This results in cheaper and easier system maintenance.

d. Additional Benefits

Additional benefits are realized by operating in an open system environment. Frequently, open systems result in the following benefits:

- Infrastructure supports distributed computing environment
- Minimal reliance on proprietary products and protocols
- Reduced costs due to increased competition
- Reduced probability of schedule delay due to increased competition
- Greater reliability due to increased test base (more users)
- Increased portability of applications due to use of standard protocols
- Increased interoperability through the use of standard protocols [Ref. 3]

III. METHODOLOGY AND TOOLS USED FOR DEVELOPING THE MCIAIS DATA ARCHITECTURE

This chapter describes the methodology advocated for the development and distribution of the data architecture for MCIAIS. It also presents an overview of the IDEF1X modeling technique used for the data modeling aspects of the MCIAIS redesign project. Additionally, a review of the CASE tool selected for this research is presented. Finally, this chapter briefly describes the features of the target DBMS used for this project and the associated procedural language.

A. DEVELOPMENT METHODOLOGY

The methodology used for developing the data architecture is a four step process. These steps vary slightly depending upon whether the architecture is being modeled for the entire business enterprise or for a business unit within the enterprise. The process of developing the data model for business units within the enterprise is illustrated in Figure 3-1. The steps of the process are: 1) list candidate data entities, 2) define the entities and their associated attributes and relationships, 3) develop the data model, and 4) relate the entities to the business processes to reveal candidate applications. These steps are further described below.

1. List Candidate Data Entities

The first step in the design of the business unit data architecture for a client/server based information system is to list the candidate data entities. This begins with the study of existing data sources and is accomplished by: 1) gathering information by distributing and collecting questionnaires, and interviewing developers, administrators, and end-users; and 2) studying existing system documentation, screens, and reports. It is important to

identify all potential entities during design in order to minimize future maintenance efforts.

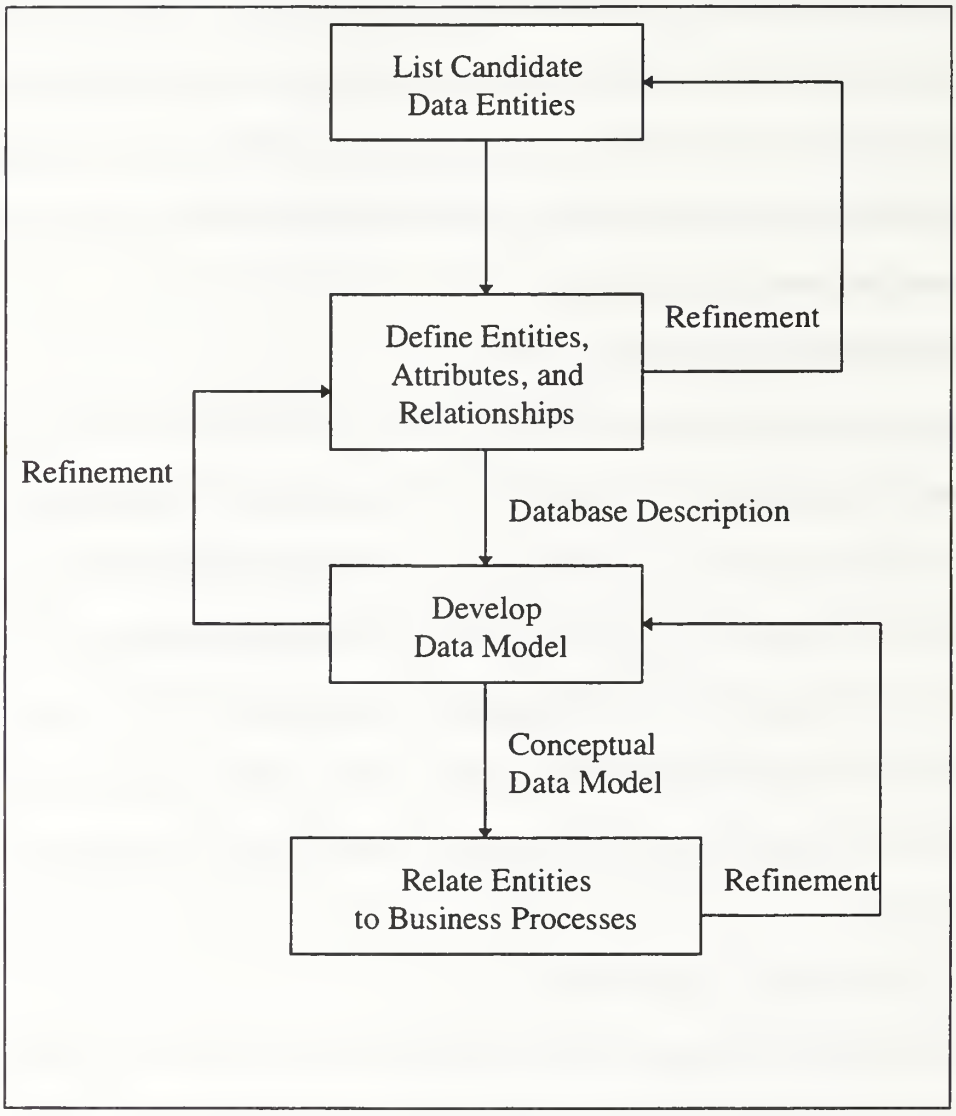

Figure 3-1. Data Model Development Methodology

a. Questionnaires and Interviews

Analysis begins with the distribution and collection of questionnaires and the conduct of interviews with developers, programmers, administrators, and users of the

current database. Interviews are especially useful. While interviews with database developers and administrators focus on the structure, functions, and modification of the existing database, user interviews focus on the applications and processes which access the data.

User interviews are particularly useful in identifying data elements used in manual processes that should be modeled in order to automate these processes. Whenever possible, onsite interviews are conducted [Ref. 10]. These interviews afford the opportunity for direct analysis of data, interviewing the largest number of key participants, and for a demonstration of the existing system. Additionally, copies of all available documentation can be easily obtained. Follow up communications is accomplished by scheduling telephonic conferences, as well as the exchange of electronic mail [Ref. 10].

b. Documentation, Input Screens, and Output Report Review

Information gathered from documentation, user input screens, and output reports augments the data collected from the developer and user interviews. All available information on existing data structures, data types and definitions, database interface and download procedures, and data process definitions is reviewed. User input screens reveal further data needs as well as the structure of existing data. Data dictionaries are reviewed. Information stored, accessed, and updated manually is examined in detail, and this information is grouped into appropriate entities in order to facilitate the automation of manual processing.

Review of existing processes is done in conjunction with business process re-engineering team members, as the development of the future data model supports their

re-engineering effort. Output reports are reviewed with end-users to determine their usefulness and any data elements contained in them which are not considered useful are noted. Additionally, users are asked to identify reports not currently available which would be desirable in the future system.

c. Develop Candidate Entity List

Based on the collected data requirements, a candidate entity list is drafted. This list forms the basis of the data model. It will be refined during subsequent development stages.

2. Define The Entities and Their Associated Attributes and Relationships

Once the candidate entities are listed, they must be clearly defined. Additionally, candidate attributes for these entities are developed and defined. Many of the candidate attributes will be identified during the analysis of the existing system. Once identified, the attributes must be defined in some detail. Finally, the relationships between the entities must be identified and defined.

It is not necessary that all attributes and relationships be listed and defined at this stage. Additional attributes and relationships will reveal themselves throughout the development process. At a minimum, primary and foreign keys must be identified during this stage. Primary and foreign keys are discussed later in this section.

3. Develop the Data Model

The most effective means of communicating the structure of a database is through the use of a graphically depicted conceptual data model. There are several commonly used modeling techniques that have been standardized and are easily understood. For this project, IDEF1X was selected as the modeling technique. IDEF1X has been standardized

by the National Institute of Standards and Technology (NIST) as documented in the Federal Information Processing Standards (FIPS) Publication 184. As such, IDEF1X is the standard modeling technique for use by the federal government, including the Department of Defense. Section B of this chapter presents a brief overview of the IDEF1X Model.

4. Map Entities to Processes in the Process Model

After the initial identification of target entities, these entities are mapped to business processes in the business process model developed by the business process re-engineering team. The business process re-engineering team groups processes by related entities. As these processes are grouped and refined, the candidate applications emerge which will be developed by the application development team. During this process, entities are refined, broken into sub-entities, and the normalization of the data model is accomplished. No entity should exist that is not created, read, updated, deleted or archived by a process identified during the business process analysis. Additionally, all automated business processes must be supported by the underlying database.

a. Map Entities to Business Processes

In this step, entities are clearly defined to the process re-engineering team. The process re-engineering team maps these entities, in the form of a CRUD matrix, to the business processes. In the CRUD matrix, a "C" designates entities that are created by a process, an "R" those entities that are read by a process, a "U" those entities that are updated by a process, a "D" those entities that are deleted by a process, and an "A" those entities that are deleted by a process. Figure 3-2 shows a portion of a CRUD matrix.

PROCESS / ENTITY	GRADING				WRITE PME				WRITE COURSE			
	C	R	U	D	C	R	U	D	C	R	U	D
ERROR_CODES		R				R				R		
ERROR_LISTING	C	R	U	D								
ESSAY_EXAM_ANSWER	C	R			C	R	U	D				
ESSAY_EXAM_QUESTIONS		R			C	R	U	D				
EXAM		R							C	R	U	D

Figure 3-2. Sample Portion of a CRUD Matrix

b. Refine the Conceptual Data Model

Throughout the development process, the conceptual data model is refined. Refinement includes the addition, modification, and deletion of entities, attributes, and relationships, including their associated definitions, in order to reflect the current business environment. As refinement occurs, the model must be continually normalized. Normalization is the process of breaking entities into two or more sub-entities to eliminate the undesirable consequence of modification anomalies. [Ref. 1]

Modification anomalies are classified as deletion or insertion anomalies. A deletion anomaly exists when, with one deletion, facts are lost about two logical entities. An insertion anomaly exists when data cannot be inserted into a logical entity. Normalization is the process of removing these anomalies. Normalization can occur at the entity level, by analyzing the data model, or at the table level, by analyzing the generated relational schema. In this discussion we assume it is conducted at the entity level. In this case, all entities are first broken down to resolve remaining "many - to - many " relationships into “one - to - many” relationships.

Suppose we have an entity in our data model as depicted in Figure 3-3. As indicated, the primary key of the student entity is the composite of two attributes: Student ID and Course. The entity also includes a third attribute: Prerequisite Course.

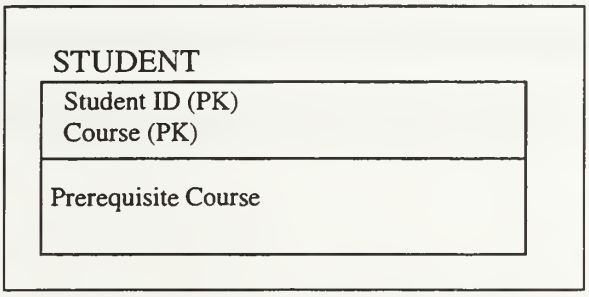

Figure 3-3. Student Entity

Information pertaining to this entity might appear in a database as shown in Table 3-1.

Student ID	Course	Prerequisite Course
10	Warfighting	None
14	Warfighting	None
33	Command and Staff	AWS
36	Command and Staff	AWS
99	Map Reading	None
99	AWS	Warfighting

Table 3-1. Student Enrollment Information

Suppose that Student 99 drops out of the AWS course If the record for Student 99 is deleted, then we lose not only the fact that Student 99 was enrolled in AWS, but also the fact that Warfighting is a prerequisite for AWS. This is an example of a deletion anomaly. Imagine that another course, JPME Phase II, is offered to students with

a prerequisite course of Command and Staff. That information cannot be entered into the database until a student enrolls in the JPME Phase II course. This is an example of an insertion anomaly.

Normalization forms are defined in terms of the type of anomalies they address. Commonly understood forms are first through fifth normal forms. For the purposes of this research, it is sufficient to describe only the first through third normal forms.

An entity is said to be in first normal form if it meets the definition of a relation. For an entity to be a relation, the attributes in the entity must be single valued, and neither repeating groups nor arrays are allowed as values. All entries for a particular attribute must be of the same data type. Each attribute must be uniquely named. Finally, no instances of a particular can be identical. Inspection reveals that the data illustrated in Table 3-1 is in first normal form.

As Table 3-1 illustrates, first normal form does not preclude modification anomalies. Table 3-1 still suffers from update anomalies. The reason this table suffers from update anomalies is that the third attribute, Prerequisite Course, is dependent upon only part of the composite key. The Prerequisite Course is not dependent upon the Student ID. It is dependent only upon the Course. The attribute is not dependent upon the whole key. An entity is said to be in second normal form if every attribute dependent on the entire key.

Consider a modification of the student entity, as shown in Figure 3-4. In this situation, a student can only take one course and a course can only have one prerequisite. Thus, Course is dependent upon Student ID, and Prerequisite Course is

dependent upon Course. In other words, the prerequisite is determined by the course, and the student's course is determined by the student's ID. Therefore, it can be said that the student's ID determines the prerequisite by the way of Course. This arrangement of dependencies is formally referred to as a transitive dependency.

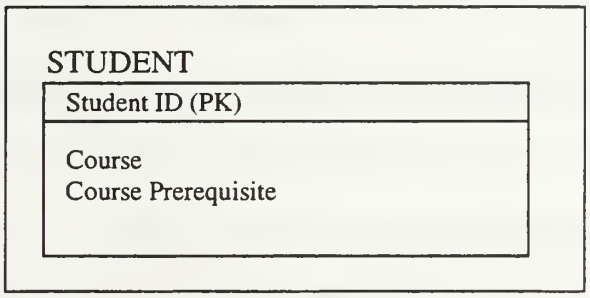

Figure 3-4. Entity in Second Normal Form

In this example, STUDENT is a relation, and its attributes are dependent upon the key, so first normal form is achieved. Course and Course Prerequisite are also dependent upon the whole key (this should be obvious, as the key is not a composite key). Thus, second normal form is also achieved. Yet modification anomalies can still exist. This is due to the transitive nature of the dependency of Course Prerequisite upon Student ID. Removing transitive dependencies will yield relationships that are in the third normal form. An entity is said to be in third normal form if every attribute is "about the key, the whole key, and nothing but the key". In the example shown in Figure 3-4, third normal form is not present, as Course Prerequisite depends not only on the primary key, but also on the Course, which is no longer part of the primary key.

It should be noted that at the enterprise level, the steps of defining a data model described above are modified slightly. These steps are: 1) list candidate data subjects, 2) define the data subjects and their associated attributes and relationships, 3) develop the

data model, and 4) relate the data subjects to the business functions in order to reveal major subsystems.

At the enterprise level entities are referred to as ***data subjects***. The collection of data subjects should encompass all of the people, places, and things for which data must be stored in the organization. Step one of the process lists these data subjects, while in step four they are mapped to business functions and clustered appropriately in order to reveal major subsystems, which can be examined in detail later for further development. Before beginning this mapping process at the enterprise level, it is useful to map the data subjects to organizational units, and to physical locations at the enterprise. These matrices greatly assist in the process of mapping data subjects to business functions.

B. IDEF1X MODEL

The IDEF1X model is a simple modeling technique for database design. Though not as common as the Entity-Relationship (E-R) diagram technique, the IDEF1X technique has the advantage of being standardized by NIST. As such, it provides a method of insuring that the design of the conceptual model is understood by all involved parties.

Unless otherwise stated, the IDEF1X modeling features used in this thesis are in compliance with FIPS PUB 184 [Ref. 11]. Background and history on the IDEF family of modeling techniques is available in that publication.

IDEF1X was developed to meet the following requirements.

- Support the development of conceptual schemas. The IDEF1X syntax supports the semantic constructs necessary in the development of a conceptual schema. A fully developed IDEF1X model has the desired characteristics of being consistent, extensible, transformable, and expandable.

- Be a coherent language. IDEF1X has simple, clean, consistent structure with distinct semantic concepts. The syntax and semantics of IDEF1X are easy to grasp, yet are robust enough to satisfy most project needs.

- Be teachable. IDEF1X is easily taught to users and customers not familiar with semantic data modeling. It can easily be understood by management information system professionals, as well as executive level supervisors, end users, application developers, and other project team members.

- Be well-tested and proven. IDEF1X is based on many years of experience with predecessor techniques, and has been tested in numerous Air Force development projects, as well as in private industry.

- Be automatable. IDEF1X diagrams can be generated automatically by many CASE tools, including ER*win*, a forth generation CASE tool marketed by Logic Works. Additionally, ER*win* has the advantage of being capable of generating SQL based conceptual schemas for a variety of target databases and database servers. These schemas can include a variety of referential integrity triggers and indexes to insure both data integrity and performance optimization.

The IDEF1X model consists of three basic constructs.

1. Things about which data is kept, e.g., people, places, ideas, events, etc., as represented by a box (these boxes are commonly referred to as entities).

2. Characteristics of those things as represented by attribute names within the box. ER*win* provides the ability to also represent, for each attribute, the associated data type and field length in the language of the target database, though this is not in accordance with FIP PUB 184. This is done to provide clarity to the model.

3. Relationships between those things, represented by lines connecting the boxes.

Entities are sets of real or abstract things such as people, places, events, or ideas. Objects in these sets share common characteristics and participate in relationships with other sets. Entities are classified as either dependent or independent, depending upon

these relationships. In IDEF1X, independent entities appear as boxes with square corners and dependent entities as boxes with rounded corners, as illustrated in Figure 3-5.

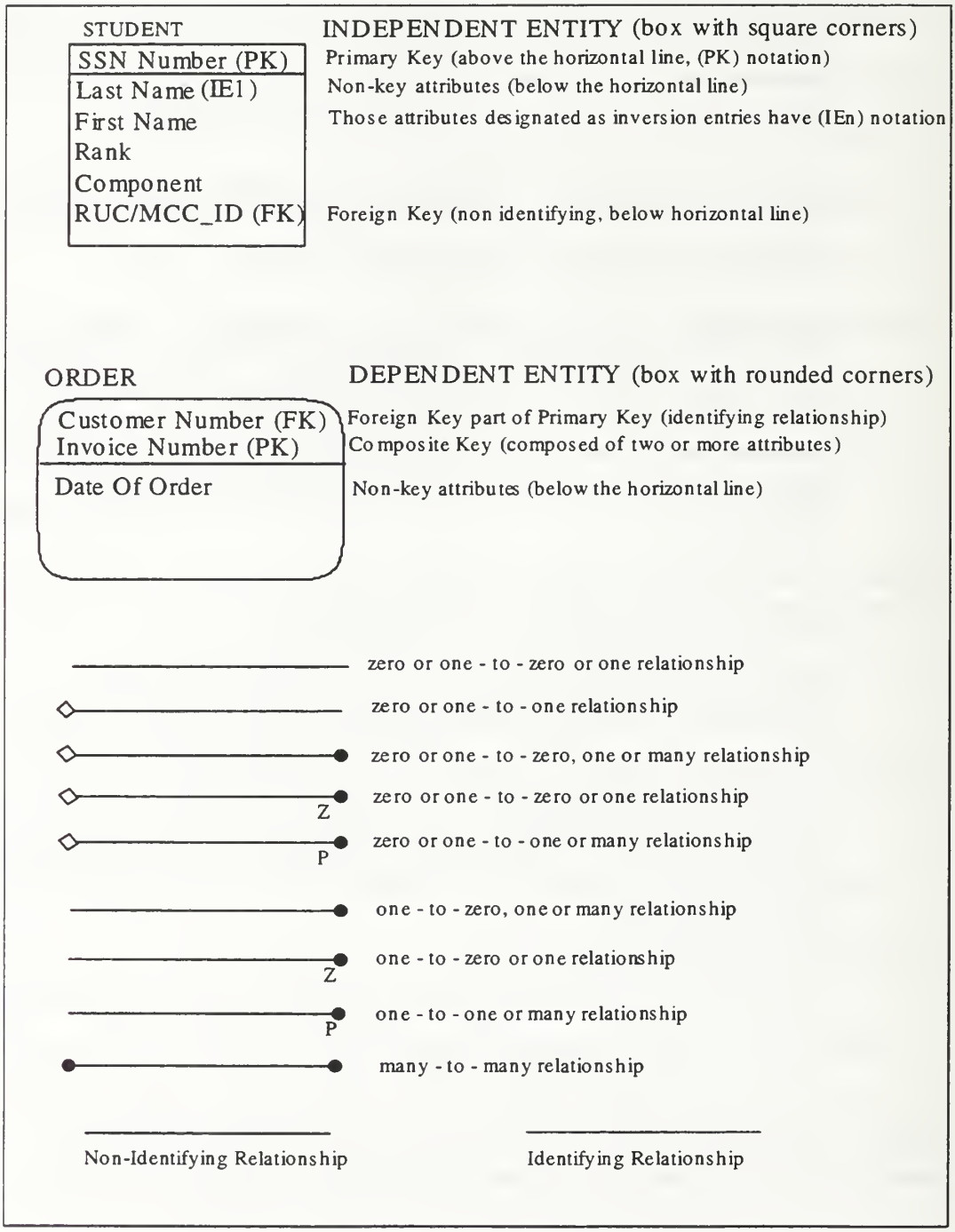

Figure 3-5. Common IDEF1X symbols

Cardinality refers to the number of instances of an entity involved in a relationship with another entity. Minimum cardinality refers to the minimum number of instances of an entity that may be involved in a relationship with another entity. Maximum cardinality refers to the maximum number of instances of an entity that may be involved in a relationship with another entity.

Using the student-course relationship from Figure 3-6 as an example, a student may take zero, one, or many courses. Similarly, a course may have zero, one or many students. In this example, zero is the minimum cardinality and many is the maximum cardinality of both student and course. Commonly, minimum cardinalities are not referred to when describing relationships. Thus, the relationship of students to courses would be described as a "many - to - many" relationship. The symbols used to represent various relationships and their cardinalities are illustrated in Figure 3-5.

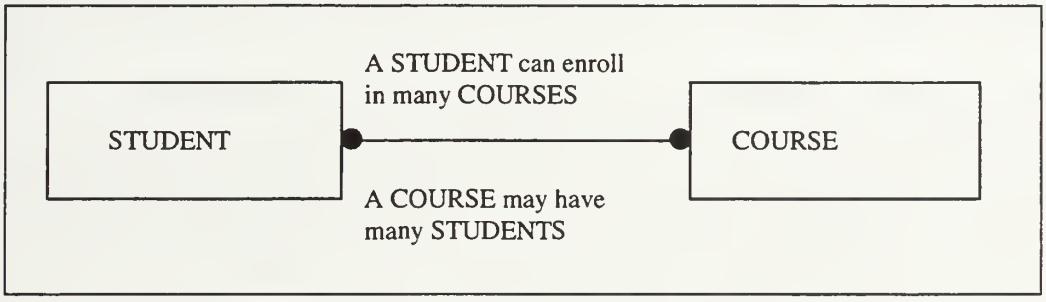

Figure 3-6. Binary Relationship

Participation constraints indicate whether the existence of an entity depends on it being related to another entity through the relationship type [Ref. 10]. In IDEF1X, entities constrained by participation are known as dependent entities. Their existence is said to depend on the existence of another entity. In IDEF1X, entities are said to be either

dependent or independent. Independent entities are said to be parents, and their dependent entities are known as children.

Additionally, some relationships involve entities which share common characteristics. These are referred to as ***generalization*** or ***inheritance*** hierarchies. They are also referred to as ***sub-type relationships***. For example, different types of PERSONNEL might exist in an organization, such as COURSE_WRITERS, CUSTOMER_SERVICE_REPS, and WAREHOUSEMEN. A generalization entity called PERSONNEL is formed to represent information common to all three types of PERSONNEL. The relationship would be expressed as, "a person may be a course writer, customer service representative, warehouseman, or other."

A generalization hierarchy may be complete or incomplete. For example, a generalization entity may exist called PERSONNEL with two sub-type entities, MALE and FEMALE. This would be a complete structure, as all personnel must be either male or female. The same generalization entity may contain the sub-type entities CONSULTANT and CIVILIAN. This structure would be incomplete, as their may be personnel who are neither consultants or civilians. Generalization Hierarchies may be appropriate in the following situations:

- The entities share a common set of attributes.
- The entities share a common set of relationships.
- The categories of an entity should be exposed (modeled as subtypes) if the business demands it, even if the categories have attributes that are different, and even if they participate in no relationships distinct from other categories.

The symbols associated with generalization hierarchies are illustrated in Figure 3-7.

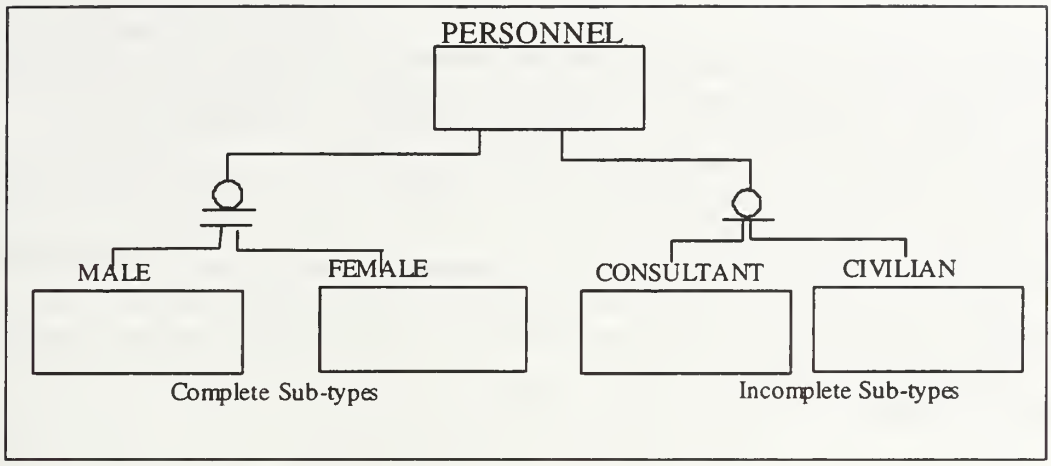

Figure 3-7. Generalization Hierarchies

Attributes contain detailed information about entities. Attributes can be single valued, such as the course number for a particular course, or multi-valued, such as the phone numbers in a particular office. Additionally, attributes may be composite (i.e., composed of several single value attributes) such as the composite attribute address, which is composed of house or apartment number, street, city, state, and zip code attributes.

An attribute, or combination of attributes, must exist for each entity which will uniquely identify an instance of that entity. This attribute (or combination of attributes) is known as the entity's***primary key***. If the primary key is a combination of attributes, it is known as a composite key. If other unique attributes exist, but are not designated as the identifying attribute, they are known as alternate keys. In the IDEF1X notation, primary keys appear above a line which horizontally bisects the entity symbol. Non-key attributes appear below the line. This is illustrated in Figure 3-5.

When two entities are related, their relationship is identified through the use of these key fields. This is done by embedding the primary key of one entity into the entity

to which it is related. For example, if there are two entities STUDENT and RUC/MCC, the relationship is clearly described as a RUC/MCC has many STUDENTS, but a STUDENT may belong to one and only one RUC/MCC. In this case the primary key from RUC/MCC (RUC/MCC_ID) would be embedded in each instance of STUDENT belonging to that RUC/MCC. These embedded keys identify which students belong to a particular RUC/MCC. They are known as ***foreign keys***.

A foreign key may be the primary key (or part of the primary key) of the entity in which it is embedded. If that is the case, the relationship is said to be an identifying relationship. If, however, the foreign key is not needed to uniquely identify the entity in which it is embedded, the relationship is said to be non-identifying. In the example above, a STUDENT is uniquely identified by his social security number (SSN_ID), and does not depend on the foreign key RUC/MCC_ID for his identity.

C. OVERVIEW OF ER*WIN*

ER*win* is a database CASE tool designed specifically for client/server system development [Ref. 12]. It combines a Windows GUI with an entity-relationship (ER) diagramming tool, provides custom editors to define logical and physical database objects, and supports many of the leading SQL database servers. As such, it facilitates the creation or re-engineering of relational databases in client/server environments.

ER*win* is one of the few available comprehensive CASE tools which supports the IDEF1X methodology. Using the IDEF1X diagramming method, ER*win* allows the user to create comprehensive data models to document complex data environments. ER*win* models can be illustrated in color to assist in readability.

1. Schema Generation

As ER*win* builds a data model, it simultaneously creates matching physical data structures and stores them in a dictionary. When the data model is completed, the user selects the target server where he wishes to build the database. The ER*win* schema generation feature is then used to generate the database schema in the target server. ER*win* builds the physical database, including all tables, indexes, stored procedures, triggers, and any other components needed to manage the data.

2. Schema Maintenance

ER*win* possesses a forward and reverse engineering capability for database schema generation. This allows ER*win* to connect to the target server's system catalog, import an existing database, and generate the corresponding logical data model. It allows the user to add, update or delete data definitions, modify relationships, and add or subtract entities and attributes. The user can then rebuild the physical database on the original or a different target server. This process is depicted in Figure 3-8.

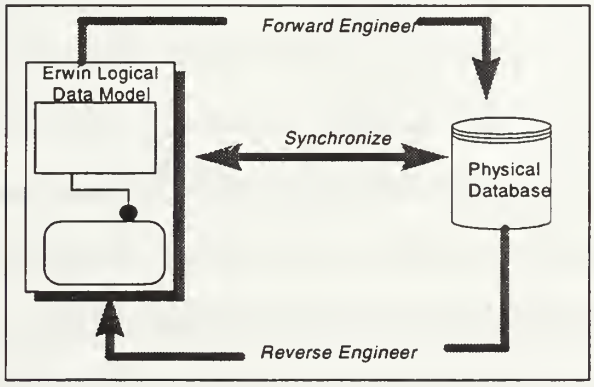

Figure 3-8. ER*win* Forward and Reverse Engineering, From Ref. [12]

When ER*win* generates the physical schema, it creates a data definition language (DDL) script using the correct SQL syntax for the target server. This code can be viewed and modified prior to database creation. ER*win* supports databases which allow for

stored procedures and triggers by providing template editors and macros which automate the creation of procedures to enforce business rules. To further ease maintenance requirements, ER*win* allows users to create tablespaces and rollback segments as well as specify the location and storage parameters for database tables from within ER*win*.

3. Server Compatibility

ER*win* is compatible with a large number of database platforms. This feature greatly enhances a users ability to migrate an existing database from one platform to another. ER*win* can be used to reverse engineer an existing database, create a logical data model, modify that model, and then regenerate the database on new target server. The current version of ER*win* is compatible with the target server environments illustrated in Table 3-2:

SQL Databases		**Desktop Databases**
• AS/400	• Progress	• Microsoft Access
• DB2/MVS	• Red Brick	• Paradox
• DB2/2	• Rdb	• dBASE III
• Informix	• SqlBase	• dBASE IV
• Ingres/OpenIngres	• SQL Server	• Microsoft FoxPro
• InterBase	• SYBASE	• Clipper
• NetWare SQL	• Teradata	
• Oracle	• Watcom/SQL Anywhere	

Table 3-2. ER*win* Compatible Databases, From Ref. [12]

4. Interface with the BP*win* Process Modeling CASE Tool

The IDEF family of modeling methodologies includes the IDEF0 methodology for process modeling. ER*win* can exchange information with BP*win*, which is Logic Works' IDEF0 business process modeling tool This feature is useful to users who are developing business process models and data models simultaneously, as was the case with the

MCIAIS redesign project. Any changes made to the entity and attribute information in either model can be merged so that the models remain consistent.

5. ER*win* ISSUES

Two issues associated with ER*win* modeling merit special attention. These issues, restricted use of "many - to - many" relationships and inversion entry notation, are discussed below.

a. Restricted Use of "Many - to - Many" Relationships

In data modeling, entities are commonly defined as items of interest in the user's environment. As such, they are usually real world objects. The version of ER*win* used for this project restricts the modeler's use of "many - to - many" relationships for the purpose of schema generation. All "many - to many" relationships must be resolved into "one - to - many" relationships in order to generate schemas using ER*win*. This forces the model to depict conceptual items, which are not real world objects, as entities.[1]

For example, a student can enroll in many courses, and a course can have many students. Both student and course are real world objects and as such, in data modeling terms, would be known as entities. Although these entities are related in a "many - to - many" relationship, as shown in Figure 3-9a, ER*win* forces the modeler to resolve this relationship into two "one - to - many" relationships by creating an "association entity" between student and course, as shown in Figure 3-9b. This entity might be called "STUDENT_COURSE_X". As a result, the IDEF1X diagram as

---

[1] The most recent release of ER*win*, version 3.0, resolves many - to - many relationships automatically.

modeled in ER*win* can easily become cluttered with these association entities, created for the sole purpose of resolving "many - to - many" relationships.

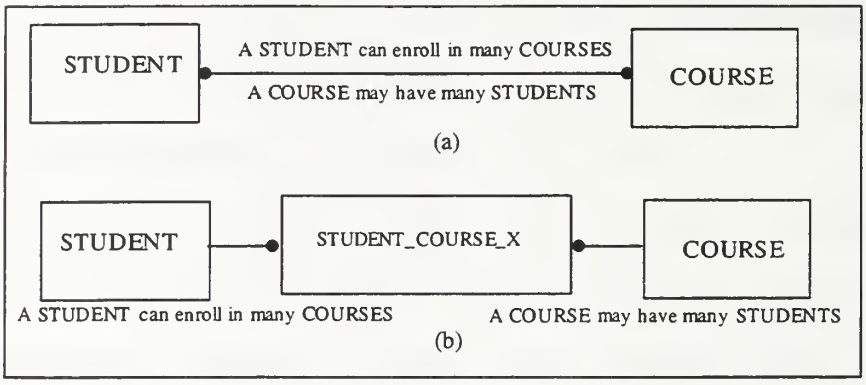

Figure 3-9. Resolved Many - to - Many Relationships

Because of this limitation, it is useful to first model the desired system without regard to this somewhat artificial constraint, thereby allowing for the existence of "many - to - many" relationships on the diagram. These relationships can later be resolved into "one - to - many" relationships for the purpose of schema generation. The enterprise-wide data model described in Chapter IV has unresolved "many - to - many" relationships.

b. Inversion Entry Notation

In ER*win*, some attributes are known as inversion entry attributes. An inversion entry represents an additional way that the user wants to access data. An inversion entry is an attribute or group of attributes which will commonly be used to an the entity. For example, consider the entity STUDENT in Figure 3-4. It may be identified by the key field, Student Social Security Number. However, the user may want to search the entity for a student whose last name is Smith. If the attribute Last Name is designated as an inversion entry, an index will be built on the attribute Last Name, and

the user will be able to rapidly access all instances of the STUDENT entity with the Last Name of Smith.

Inversion entry notation is not standardized in FIBS PUB 184, but is a key feature in ER*win*. When inversion entries are designated in an ER*win* model, they cause an index to be built on those designations whenever a schema is generated for a target database. Because of this utility, they are used to designate those fields which, although not unique, are most likely to be searched during database queries.

D. OVERVIEW OF ORACLE 7 AND PL/SQL

Oracle has emerged as the DoD standard database server in recent years. Additionally, it was initially selected as the target platform for the future MCIAIS system by MCI. As such, Oracle was chosen as the target database for the prototype designed for MCI as part of this case study.

A version of Oracle, known as Personal 7, was used to develop the prototype. This version has most of the features found in the more powerful Oracle 7 server, but is able to run on a personal computer. It is considered fully compliant with the client/server model, as the database is logically and physically separated from the applications, although both reside on the same machine.

1. Oracle 7 Architecture

Most SQL database servers are designed around one of three server architectures: process-per-client, multithreaded, and hybrid [Ref. 5]. Oracle 7 uses the hybrid architecture. A brief discussion of all three approaches facilitates understanding of the Oracle 7 architecture.

a. Process-per-client Architecture

Process-per-client is the most protected, stable and secure database server architecture. This stability is referred to as maximum bullet-proofing [Ref. 5]. The database runs in one or more background processes on the server. Each client is given its own process address space on the server. This architecture has the advantage of protecting users from each other while protecting the database manager from the users. Processes are each assigned to an individual processor on a multiprocessor server. However, this architecture normally consumes more memory and CPU resources than alternative solutions. Therefore, it can bog down if not managed effectively. Transaction processing (TP) monitors can increase efficiency by tagging those processes which are reusable [Ref. 5]. Figure 3-10 depicts this architecture.

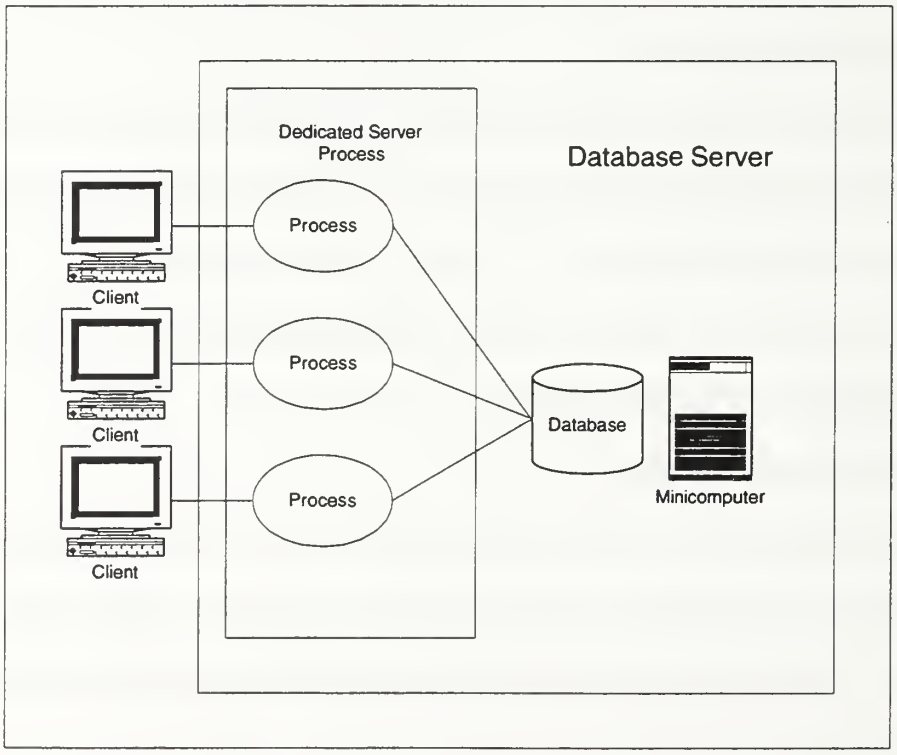

Figure 3-10. Process-per-client Database Server Architecture, After Ref. [5]

Examples of the Process-per-client Database Server Architecture include DB2/2, InterBase, Informix, and Oracle 6.

b. Multithreaded Architecture

Multithreaded architectures provide the most efficient performance by running all user connections, applications, and the database in the same address space. An internal scheduler on the server allocates CPU time, thereby eliminating the local OS from scheduling algorithms. This conserves memory and CPU cycles. This architecture is more open than process-per-client, in that the server implementations are cross-platform portable as they do not require many local OS services. This architecture is illustrated in Figure 3-11.

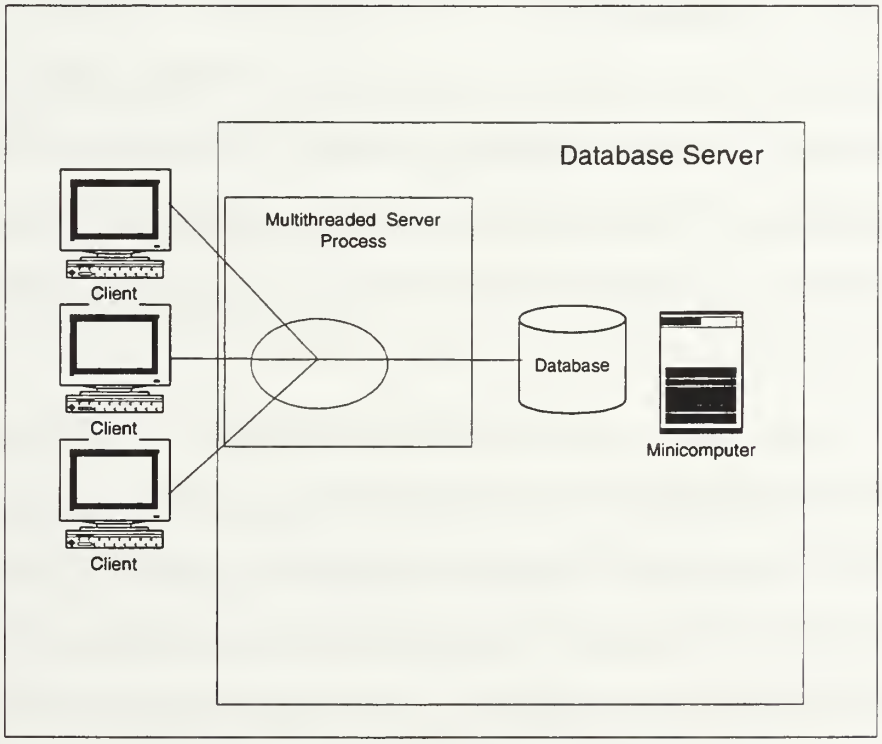

Figure 3-11. Multithreaded Database Server Architecture, After Ref. [5]

Multithreaded systems are prone to deadlock conditions and other failures, as errant processes interfere with competing processes in the same address space. Finally, long duration tasks monopolize CPU time. Examples include Sybase and SQL Server.

c. Hybrid Architecture

Hybrid architectures are relatively new and consist of three components, as depicted in Figure 3-12:

- Multithreaded network listeners which assign clients to dispatchers during the initial connection.
- Dispatchers which place messages in an internal queue.
- Reusable shared server processes that take work off the queue, execute the specified tasks, and place their responses in a separate out queue for dispatch to the corresponding client.

This architecture provides a protected environment for each user task, without requiring each client to have a permanently assigned process address space. However, it suffers from queue latencies due to the queuing required during requests for services and service response. These queues can interfere with the TP monitor's scheduling algorithms [Ref. 5].

Oracle 7 was the first database server to implement the hybrid architecture. Performance with this architecture can be expected to range from twenty percent improvement to a twenty percent degradation in comparison with previous architectures [Ref. 5]. This range of performance is dependent upon the effectiveness of the TP monitor, the scheduling algorithms, and the degree of queuing latency, as well as the nature and number of processes competing for limited resources.

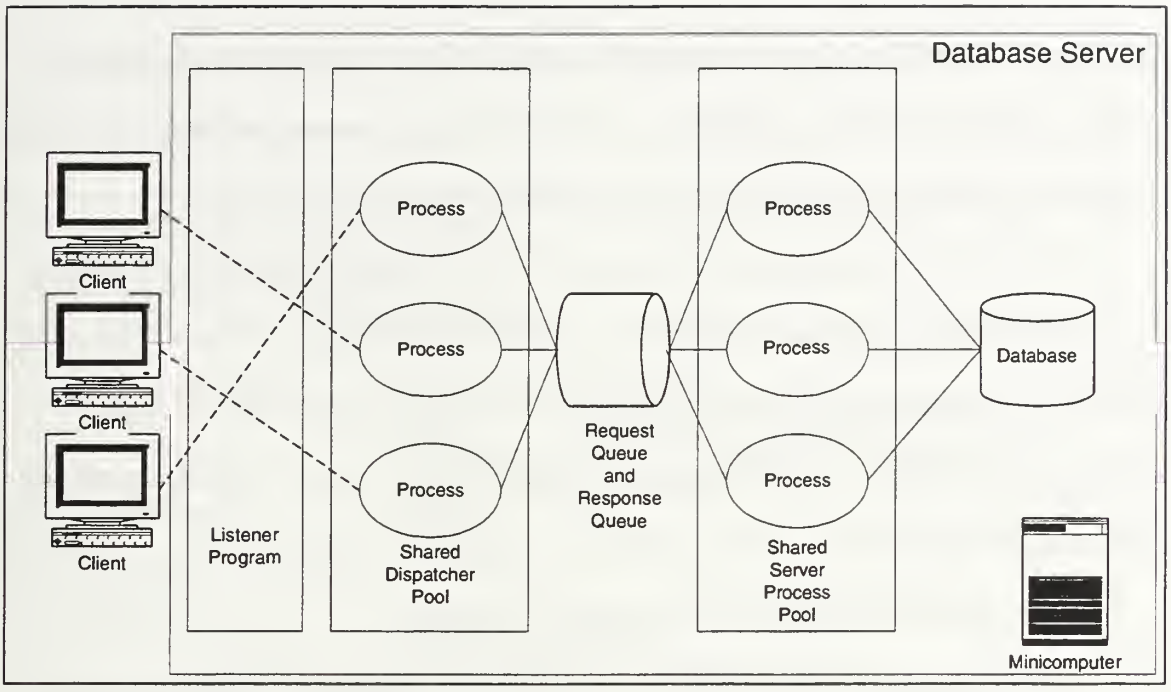

Figure 3-12. Hybrid Database Server Architecture, After Ref. [5]

2. PL/SQL Overview

PL/SQL, which stands for "Procedural Language extensions to SQL", is not a standalone procedural computer language. Rather, it ships with other software products as a tool to enable the wrapping of SQL statements in a procedural programming language. The version of PL/SQL used in this case study was packaged with the Oracle application development tool, Oracle Developer 2000.

PL/SQL can be used to write stored procedures which are maintained on the server, as well as underlying GUI applications on the client side, such as Oracle Forms. As such, it is the underlying procedural language for Oracle's suite of development tools.

PL/SQL is closely integrated into the SQL language, yet adds programming constructs that are not inherent in the standard SQL language [Ref. 13]. Thus, PL/SQL allows the programmer to combine standard SQL statements with typical procedural

constructs. PL/SQL programs can look like traditional third generation modules, but can also include calls to SQL statements, cycle data through cursors, and employ PL/SQL tables to temporarily store data for validation against various business rules.

PL/SQL is a block-structured language. As such, each of the basic programming units used to build an application consists of a logical unit of work. This allows for both modularization and scoping of blocks. Modularization refers to the ability to separate blocks into logically independent, loosely coupled modules. Scoping refers to the ability to group logically related concepts or objects in a single block. [Ref. 13]

Each PL/SQL block has up to four different sections:

- Header section: This is relevant for named blocks only. Anonymous blocks, which cannot be called by other blocks, do not contain headers.
- Declaration section: In the declaration, variables are declared which are referenced in the execution and exception section. Variables are normally initialized in this section as well.
- Execution section: This section contains executable PL/SQL code. This code is executed by the PL/SQL run-time engine.
- Exception section: This section contains exception handlers which handle exceptions occurring during processing.

The example in Figure 3-13 shows a single program which can either enroll or disenroll a student from a course. It is a named block program and contains all four of the sections described above: Header, declaration section, execution section and exception section.

Because PL/SQL is used both in the database for stored procedures and triggers, and within the application code, the same programming language can be used for both client-side and server-side development. This is a very useful feature, as triggers and

procedures can be moved back and forth between the server and the clients to fine tune the efficiency of the system.

```
        PROCEDURE maintain_enrollments
        (action_in IN VARCHAR2 := Null,
        Crs_ID_in IN VARCHAR2 := Null,
        StudSSN_ID_in VARCHAR2 := Null)
IS
        CrsEnrollDate_V         Date :=         SYSDATE;
        CrsCCD_V                Date :=         ADD_MONTHS(SYSDATE, 48);
        SC_PME_V                VARCHAR2 :=     SEARCH_CRS_PRG_X(Crs_ID_in);
        SC_COMP_V               VARCHAR2 :=     SEARCH_STUD_COMP (StudSSN_ID_in);
        Disenrolled_V           CHAR :=         'N';
BEGIN
        IF action_in = 'DELETE'
        THEN
                DELETE FROM STUD_CRS_X WHERE
                Crs_ID = Crs_ID_in AND StudSSN_ID = StudSSN_ID_in;
        ELSIF action_in = 'INSERT'
        THEN
INSERT INTO STUD_CRS_X (Crs_ID, StudSSN_ID, CrsEnrollDate, CrsCCD, SC_PME, SC_COMP,
Disenrolled)
VALUES (Crs_ID_in, StudSSN_ID_in, CrsEnrollDate_V, CrsCCD_V, SC_PME_V, SC_COMP_V,
Disenrolled_V);
        END IF;

EXCEPTION
        WHEN DUP_VAL_ON_INDEX
        THEN
                DBMS_OUTPUT.PUT_LINE
                        (' Record already exists.  Duplicate records cannot be inserted.');
        WHEN OTHERS
        THEN
                Null;
END;
```

Figure 3-13. PL/SQL Procedure Example

For instance, a function which was originally coded for a single client may be moved to the server if additional client privileges are expanded to include that specific function. This transition from client-side to server-side can be accomplished with a few

simple mouse clicks on one of several Oracle development tools; it is not necessary to modify the PL/SQL code or any of the programs that may call the function.

E. AD HOC VERSUS FORMALLY DESIGNED DATABASES

Most database design projects are based on some existing data in some form. In these cases, it is important to differentiate between ad hoc designed databases and formally designed databases. These distinctions are discussed below.

1. Ad Hoc Designed Databases

Ad hoc databases are those databases which have been designed and/or implemented without the benefit of a conceptual data model. In most, but not all, cases these databases are non-relational. Usually they consist of flat file data files, with a great deal of data redundancy, and no defined relationships. Data manipulation, retrieval, and association is done programatically, and is not based on the data definitions or meta-data contained in the database. Ad hoc query language such as SQL cannot be used, and retrieval of data is often impossible.

When developing a conceptual model for a future system based on an existing ad hoc database, those portions of the ad hoc database that are relevant to the project must be re-engineered in a top down fashion, based on a good deal of investigative work. This investigative work consists of gathering facts about the data and its representations through a study of the database structure files, input forms, output reports, interviews, user screens, and all other available documentation.

2. Formally Designed Databases

Databases which have been formally developed using a conceptual data model are considered formally designed [Ref. 10]. These databases are structured relationally,

generally are developed using CASE tools such as ER*win*, and contain detailed data dictionaries and design documentation. An understanding of formal database design is crucial for the maintenance of the future system. In order to maintain the future database, and its accompanying data model, it is necessary that database maintenance personnel clearly understand the principles of IDEF1X modeling, as well as the underlying theory of relational database design.

IV. PROPOSED RELATIONAL DATA MODEL FOR MCIAIS

This chapter presents the proposed data model for MCIAIS. It begins by describing the development of the enterprise-wide data model for MCI. The process of collecting enterprise-wide data requirements, and the representation of these requirements as data subjects, attributes and relationships is discussed. Planning matrices, which are used to relate data subjects to functions in order to expose major subsystems, are explained.

The chapter then discusses the development of the MCIAIS data architecture for SSD and MIS. Development of this architecture, derived from the enterprise-wide model, follows the methodology described in Chapter III. The process of developing data requirements for SSD and MIS, and translating these requirements into entities, attributes and relationships is covered. Next, the IDEF1X conceptual data model for SSD and MIS is discussed. Next, the process of mapping entities to business processes in order to reveal candidate applications is explained. Finally, refinement of the data model is discussed.

A. ENTERPRISE DATA MODEL FOR MCI

The focus of this thesis is the development of a data architecture to support SSD at MCI. However, to facilitate an understanding of the general scope of MCI's future information system needs, it was necessary to first develop a high level, enterprise-wide data model. This analysis provided a framework for the subsequent development of the SSD and MIS data model.

1. List Candidate Data Subjects

The first step in the development of the data architecture for the future MCI client/server based information system was to study the existing databases in order to discover candidate entities. These entities, at the enterprise-wide level, are referred to in this report as ***data subjects.*** As outlined in Chapter III, this process was accomplished by 1) interviewing developers, administrators, and end-users, and 2) studying existing system documentation, screens, and reports. Normally, questionnaires are also used to analyze database requirements. A data collection questionnaire was not used however for this project because of the limited number and availability of both database administrators and end-users. Areas normally addressed in a questionnaire were covered through interviews.

a. Interviews

Data requirements analysis began by conducting interviews with developers, programmers, administrators, and users of the current database. Interviews with Mr. Joseph Rudd, the MCIAIS Administrator, and Major Donna Gerlaugh, the MCI MIS Chief, were especially valuable, as they are intimately familiar with the structure, functions, nature, modifications, and patches associated with the current flat file system and its related programs. Interviews with the users, especially in SSD, were useful in identifying the underlying purposes of the data, as well as their associated processes and business functions.

Whenever possible, on-site interviews were conducted at MCI. On-site interviews afforded the opportunity for direct analysis of data as well as interviewing the largest number of key participants, and allowed for a demonstration of the existing

system. Additionally, copies of all available documentation were readily available. Follow up communication was accomplished by scheduling telephonic conferences with Mr. Rudd and Major Gerlaugh, as well as through the exchange of electronic mail.

b. Documentation, Input Screens, and Output Report Review

Information gathered from documentation, user input screens, and output reports was used to augment the data collected from the developer and user interviews. In the case of MCI, the existing data dictionary is incomplete, poorly documented, and out of date. Read ahead material provided by MCI did not reflect current data stores and data definitions in many instances. The output reports provided by MCI for analysis were of limited use. No systematic analysis of existing output reports was conducted, as no user interviewed placed any importance on the existing reports. The common theme seemed to be that more useful reports were needed, based on statistical analysis not currently conducted.

c. Develop Candidate Data Subject List

The information gathered through interviews and inspection of documentation was used to develop the candidate data subject list for the enterprise-wide MCI data model. This list was developed in conjunction with the process re-engineering team, as their work to identify current and future business functions frequently resulted in the identification of candidate data subjects. Examination of the databases currently maintained by MCIAIS was especially helpful in identifying candidate data subjects for existing processes. A total of 26 data subjects were identified.

The most obvious data subjects, based on the mission of MCI, are COURSE and STUDENT. MCI business rules further define courses as being either

professional military education (PME) programs or military occupational specialty (MOS) related courses. The former were identified by the data subject PROGRAM and the latter by the data subject COURSE. These data subjects, STUDENT, PROGRAM, and COURSE, can be identified through direct examination of the MCIDB database. Further examination of existing databases uncovered additional candidate data subjects. For example, examination of the MMSD and RMPDB databases revealed the data subject MCTF_PERS.

Examination of the MSEXAM database revealed the existence of the data subject EXAM, and also identified the need to develop candidate data subjects to support the logistics functions at MCI. Examination of these logistics functions, such as ordering, receiving, storing, inventorying, packaging, delivering and disposing of material, resulted in the development of several additional candidate data subjects. These include INVENTORY, ORDER, PURCHASE, and WAREHOUSE. Accurate modeling of these logistics related data subjects necessitated a tour of the MCI warehouse during the initial site visit conducted in August 1996. This visit revealed an additional item of material being developed, stored, and distributed by MCI: MOS related job-aids. This resulted in the creation of the candidate data subject JOB_AID.

During the warehouse tour, it was discovered that all material developed, maintained, issued, and evaluated by MCI, including courses, programs, job-aids, and exams, consisted of a large number of basic components. Many of these components, such as envelopes, were shared by several courses, programs, job-aids, and exams. As a result, the candidate data subject COMPONENTS was identified and added to the candidate data subject list.

Interviews with course developers uncovered several additional candidate data subjects. The current MCI business rules divide course development into two groups: The Occupational Specialty Department develops MOS related courses and job-aids, while the Professional Military Education Department develops PME programs. In the future, MCI hopes to maintain all course, job-aid and program copy material in a central database. In order to support this goal, the data subjects of COPY_MATERIAL, COURSE_DEVELOPERS and PROGRAM_DEVELOPERS were listed. Additionally, COPY_MATERIAL was divided into the sub-types of JOB_AID_COPY_MATERIAL, COURSE_COPY_MATERIAL, AND PROGRAM_COPY_MATERIAL.

Interviews with SSD personnel resulted in the identification of several data subjects designed to support the customer service business function. Currently, SSD clerks provide telephonic support to a wide variety of customer inquiries. In the future, MCI hopes to develop a robust customer help desk supported by an underlying database. The data subjects of SSD_PERSONNEL, CUSTOMER, and ISSUE_COMPLAINT were added to support the customer service function.

Additional interviews conducted during the initial August 1996 visit to MCI revealed the remaining candidate data subjects. The data subject MCI_PERSONNEL was created to capture administrative information on all personnel, both civilian and military, stationed at MCI. A discussion of personnel issues uncovered two previously unmentioned business functions: Training and support of military parades by military personnel. This resulted in developing TRAINING and EVENTS as candidate data subjects. Discussion of the budgeting function resulted in the creation of the FINANCIAL data subject. Support of the advertising function dictated the creation of

the data subject ADVERTISEMENT. Finally, it was important that MIS personnel be able to track all information systems related equipment belonging to MCI, so the data subject IS_EQUIPMENT_INVENTORY was created. A total of 26 candidate data subjects were developed and are shown in Appendix B.

2. Define The Data Subjects and Their Associated Attributes and Relationships

From the list of 26 candidate data subjects, definitions of enterprise-wide data subjects were developed. These definitions are shown in Appendix C. Clearly defining these data subjects was vitally important to facilitate further development of both the conceptual data architecture and the process architecture, as well as the application prototype. It was critical that all personnel involved in the project agree on the definitions, in order to prevent inconsistencies between the architectures and the associated prototype.

Once the data subjects were clearly defined, primary keys were identified, as shown in Appendix D. Initially, all candidate keys for each data subject were listed. Primary keys were selected from the list of candidate keys. In some instances, selection of keys was dictated by the presence of attributes in the current databases. Selection of existing attributes as keys was done in order to ease data migration constraints. For example, because social security numbers on all students exist in the current database, this was a logical choice as the primary key for the STUDENT data subject.

Following primary key selection, additional candidate attributes were identified for each data subject, as illustrated in Appendix E. These attributes were then defined as shown in Appendix F. Finally, the relationships between the data subjects were

identified and defined as shown in Appendix G. As was the case with data subject definitions, clearly defining these attributes and the relationships between data subjects was important to insure consistencies between the data architecture, process architecture, and application prototype.

3. Develop an Initial Conceptual Data Model

An enterprise-wide data model, containing the 26 primary subject areas and their relationships, was developed in order to provide a better conceptual understanding of the architecture needs at MCI in terms of development costs, and future hardware, software, and peopleware needs. This enterprise-wide, high level data model is included as Appendix H. Before proceeding to development of the data model for SSD, the enterprise-wide data model was verified with MCI database managers and SSD supervisors.

4. Development of Planning Matrices

The enterprise-wide data model, containing the twenty-six primary data subject areas and their relationships, was provided to the process re-engineering team, and was used to map data subjects to functions in the process model. Before mapping the data subjects to the business functions, data subjects were mapped to organizational units, and to physical locations at MCI. Following the development of these matrices, data subjects were mapped to the functions in the business process model. This mapping was used to develop a CRUD matrix, and assisted the process re-engineering team in the identification of all the major subsystems within the MCI enterprise, as well as in the identification of the major subsystems that should logically be included in the further

refined SSD and MIS process model. [Ref. 14] These matrices are included as a series of exhibits in Appendix I.

Matrix	Exhibit #, Appendix I
Data Subjects vs. Organizational Units	Exhibit 1, Appendix I
Data Subjects vs. Physical Locations	Exhibit 2, Appendix I
Data Subjects vs. Functions (CRUD Matrix)	Exhibit 3, Appendix I

Table 4-1. Matrices Used to Map Data Subjects to Functions

B. MCIAIS DATA ARCHITECTURE FOR SSD AND MIS

Following the development of the enterprise-wide data model for MCI, the SSD and MIS data architecture for MCIAIS was developed. The development of this architecture followed the methodology described in Chapter III and used in the development of the enterprise-wide model as described in Section A of this chapter.

1. Entities, Attributes, and Relationships

This section describes the process used for selecting the entities, attributes, and relationships of the SSD model. This process conforms to steps one and two of the methodology described in Chapter III.

a. List Candidate Data Entities

Development of the SSD architecture began with an examination of the 26 candidate data subjects developed for the enterprise-wide data model. From this list, an initial list of candidate entities for the SSD architecture was formulated. This list, restricted to data subjects from the enterprise-wide model which were related to the business functions performed by SSD and MIS in support of student services, included the following entities:

1. COMPONENTS
2. COURSE
3. CUSTOMER
4. EXAM
5. JOB_AID
6. MCTF_PERS
7. PROGRAM
8. PURCHASE
9. STUDENT
10. SSD_PERSONNEL

All other data subjects from the enterprise-wide model were not considered further, as they related to business functions outside the scope of SSD and MIS. These ten entities, as well as all additional candidate entities, were classified as either main entities, domain entities, or dependent entities. Main entities represent major sets of real or abstract data. Domain entities represent entities whose members define the domain of an attribute contained in either a main or dependent entity. Dependent entities represent sets of data whose existence depends on a main entity. Two of the data subjects were renamed in order to provide more descriptive names: 1) PURCHASE became INVOICE; and 2) SSD_PERSONNEL became SSD_CLERK.

Of the ten data subjects selected from the enterprise model, eight are considered main entities, as they are independent entities and represent major sets of real or abstract things such as people, places, events, or ideas about which SSD must maintain data. The list of main entities includes:

1. COMPONENTS
2. COURSE
3. CUSTOMER
4. INVOICE
5. JOB_AID
6. PROGRAM
7. SSD_CLERK
8. STUDENT

From the enterprise model, the two remaining data subjects related to SSD functions were not considered main entities. EXAM is a dependent entity, as each instance of exam depends on a specific course. MCTF_PERS is considered to be a domain entity, as its sole purpose is to provide data with which to populate the STUDENT entity, such as name, rank, and service component.

Analysis continued with a more detailed study of the existing data sources. Additionally, follow-up interviews were conducted with MCI personnel during January 1997, February 1997, and May 1997. These interviews provided a more refined understanding of data requirements. As was the case in the enterprise level data modeling phase, interviews with Mr. Rudd and Major Gerlaugh were particularly valuable. Interviews with the SSD clerks and supervisors provided a better understanding of the business functions associated with the existing data. They also revealed limitations in the current data in terms of supporting future business functions, such as a customer help desk.

The structure of the existing system revealed a moderate degree of redundancy in current data stores. Candidate entities were developed with the goal of generating a database that eliminates data redundancy. Additionally, entities were developed to support the re-engineered business processes at MCI. During development, two additional main entities not modeled at the enterprise-wide level were discovered: 1) TRNG_NCO_MATERIAL, which is a set of materials ordered by unit training officers; and 2) ERR_LIST, which is a list of all errors recorded during the scanning of automated examination forms. Thus, the list of main entities includes:

1. COMPONENTS
2. COURSE
3. CUSTOMER
4. ERR_LIST
5. INVOICE
6. JOB_AID
7. PROGRAM
8. SSD_CLERK
9. STUDENT
10. TRNG_NCO_MATERIAL

Analysis of the existing databases uncovered a number of existing attributes which were based on abbreviations and codes whose definitions were not included in the current database. These were considered to be strong candidates for domain entities. Rather than simply restrict certain attributes to internally defined domains, it was decided that these attributes should have defined relationships with domain entities. Each domain entity serves a two-fold purpose: 1) Restrict the related attribute's domain to a finite list of values and 2) provide a stored and accessible definition for each of those finite values. Domain entities were developed to support all of the many coded fields which exist in MCIAIS and for which codes are not defined in the various databases. These domain entities include:

1. ANS_CODE
2. CAT_CODE
3. ED_LEV
4. ERR_CODE
5. GRADE
6. MCTF_PERS
7. RANK
8. REM_METH
9. RUC
10. STAT_CODE
11. STATE
12. SVC_COMP
13. TRANS_CODE

In addition to the main entities and domain entities described above, a number of dependent entities, including EXAM, were developed. All of these entities depended on the existence of one or more of the main entities. The final list of candidate entities includes 45 entities. Of these, ten are main entities and 13 are domain entities. The remaining 22 are dependent entities. Many of these dependent entities are association entities which were created during the refinement of the data model, as described below. The complete candidate entity list is included as Appendix J.

b. Define The Entities and Their Associated Attributes and Relationships

After the identification of each candidate entity, a detailed definition of each entity, its attributes, and its relationship with other entities was developed. As was the case in the enterprise-wide model, it was crucial that these definitions be understood by the process re-engineering team, as well as the prototype application development team.

Entity definitions are shown in Appendix K. Primary keys were identified, as shown in Appendix L. Candidate attributes were then identified, as illustrated in Appendix M. A total of two-hundred and eighteen candidate attributes were developed. These attributes were then defined as shown in Appendix N. Finally, the relationships between the data entities were identified and defined. This is included as Appendix O.

2. IDEF1X Model

This section describes the process used for developing the data model. This process conforms to steps three and four of the methodology described in Chapter III.

a. Develop the Data Model

A data model, containing the 45 entities and their relationships, was developed in order to provide a graphical picture of SSD data requirements and solicit feedback from users. This data model is included as Appendix P. The data model was developed in IDEF1X using ER*win*, which facilitated the rapid development of the data model. It is recommended that MCI maintain the data model using ER*win*. It is one of the few available CASE tools that supports both the IDEF1X methodology and the ability to synchronize the model with the database schema via the CASE tool graphical user interface.

b. Map Entities to Processes in the Process Model

After the development of the data model, the candidate entities, attributes and relationships, along with their definitions, were provided to the process re-engineering team. As described in Chapter I, this research was closely coordinated with this team, in order to provide a comprehensive and well integrated architecture plan to MCI for future implementation. The process re-engineering team mapped the 45 candidate entities and their associated attributes to processes described in the business process model. Initial coordination was conducted by linking ER*win* to BP*win* and exporting the data structures from the data model to the process modeling data repository.

The purpose of this step was to insure that no entity exists that is not created, read, updated, deleted or archived by a process identified during the business process analysis. Additionally, all automated business processes must be supported by the underlying database. The SSD data model, containing the 45 entities and their relationships, was used by the business process re-engineering team to map entities to

business processes in their process model. This mapping process is further discussed in *The Marine Corps Institute Information System Redesign Project Final Report*, which includes a CRUD diagram mapping the 45 entities to business processes [Ref. 15].

c. Data Model Refinement

Once the initial data model was developed for SSD and mapped to the re-engineered processes by the process re-engineering team, the existing model was continually refined based on coordination with the process re-engineering team, the prototype application developer, and the users at MCI. These continual refinements indicate that the data model is a living document which demands careful documentation. This refinement process, as well as its associated documentation, was greatly enhanced by the functionality incorporated in ER*win*.

Conversion of the IDEF1X model to a relational design was dependent upon the removal of all many - to - many relationships within the model. This was necessary as ER*win* does not automatically convert many - to - many relationships to the corresponding relational tables. Thus, for every table required in the proposed relational design, a corresponding entity must exist in the IDEF1X model. For example, consider the relationship between STUDENT and COURSE shown in Figure 4-1.

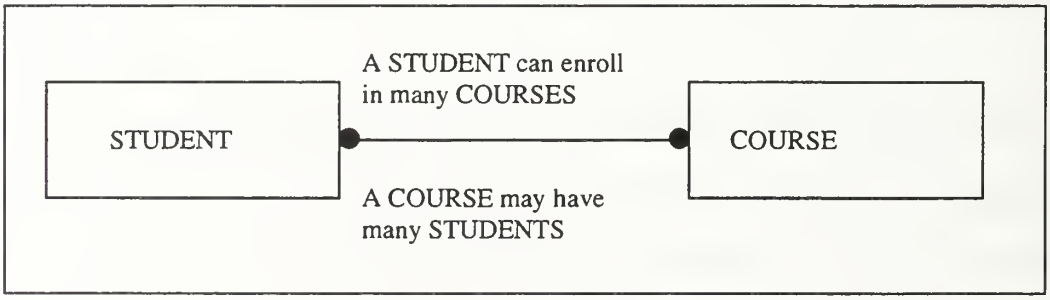

Figure 4-1. Many - to - Many Relationship

In the example, a STUDENT can enroll in many COURSES and a COURSE may have many STUDENTS. In order to convert this portion of the model to a relational design, three tables are required: STUDENT, COURSE, and an intersection table which captures the relationship between STUDENT and COURSE. Because ER*win* does not automatically resolve this many - to - many relationship into the three corresponding tables, it is necessary to refine the IDEF1X model prior to schema generation. The refinement of this many - to - many relationship into two one - to - many relationships is shown in Figure 4-2. In Figure 4-2a, the unresolved many - to - many relationship appears. In Figure 4-2b, the relationship has been resolved into two one - to - many relationships. This was accomplished by creating an association entity, STUDENT_COURSE_X, which relates STUDENT to COURSE. The primary key of this entity consists of a combination of the primary key of STUDENT and the primary key of COURSE. This association entity maps directly to a table, known as an intersection table, in the relational schema. This intersection table represents the intersection of two independent entities, STUDENT and COURSE.

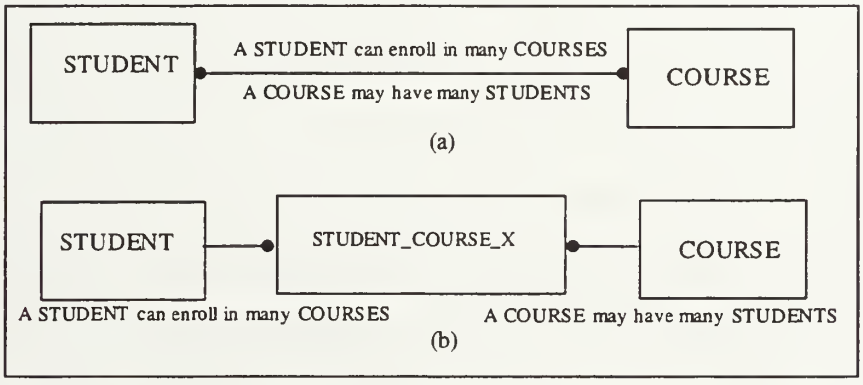

Figure 4-2. Resolved Many - to - Many Relationship

Refinement at this stage also included normalization of the data model. Once all many - to -many relationships were resolved, the remaining entities were analyzed to determine what level of normalization existed. The goal was to achieve third normal form throughout the conceptual model. Some level of de-normalization was allowed. For example, a student's rank is determined not only by his social security number (Rank depends on StudSSN_ID), but also by his grade and service component (Rank depends on Grade and ServComp), which are not part of the primary key of the STUDENT entity. Decisions to leave this level of de-normalization were based on easing the programming burden on the application development team.

Finally, ER*win* allows for the careful maintenance of the data model as well as the associated Oracle 7 schema which had been generated for use by the application developer. As the model is continually refined, both the model and its associated schema remain synchronized through the use of the synch function included in ER*win*. This process of maintenance and data model refinement is further discussed in Chapter V.

V. PROTOTYPE RELATIONAL SCHEMA

This Chapter discusses the prototype relational schema generated from the conceptual entity-relationship model and its associated triggers. This discussion focuses on the process of converting the conceptual IDEF1X model to a logical relational design schema and the benefits of utilizing server side triggers to enforce business rules. It concludes with a discussion of the maintenance requirements associated with the target system.

A. GENERATED ORACLE SCHEMA

This section discusses the generation of the Oracle schema from the developed IDEF1X data model. Oracle was chosen as the target DBMS for two reasons: 1) It is representative of an industrial-strength modern DBMS; and 2) MCI plans to eventually migrate their existing data to an Oracle database server. The development of this schema is intended to: 1) serve as a proof of concept for MCI; 2) support the application prototype developed by the application development team; and 3) illustrate the functionality of a client-server based relational database system.

As previously described, prior to schema generation, all unresolved many - to - many relationships in the IDEF1X model were resolved into a pair of one - to - many relationships in order to facilitate the use of ER*win* for schema generation. During the schema generation stage, it was discovered through trial and error that this included the modification of several recursive relationships, as illustrated in Figure 4-3. In this example, the many - to - many relationship is recursive. The relationship can be stated as such: "A program can have many programs as prerequisites, and a prerequisite program can have many associated programs." ER*win* requires this relationship to be resolved.

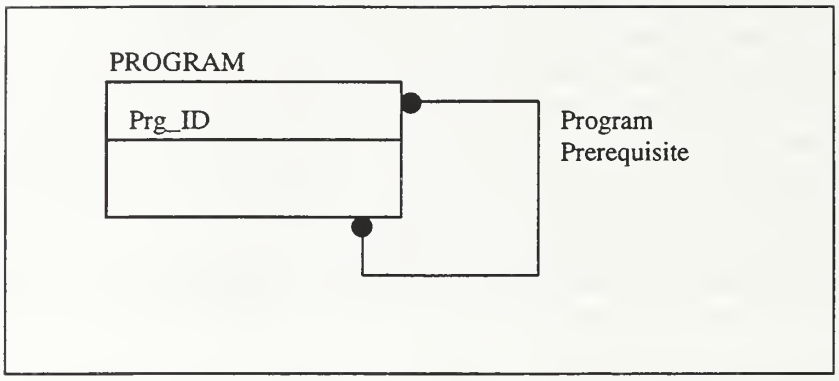

Figure 4-3. Recursive Relationship

In Figure 4-4, the recursive relationship has been resolved into a pair of one - to - many relationships through the creation of an association entity, PRG_PREREQ. This modification to the IDEF1X model must occur prior to generation of the schema with ER*win*, regardless of the target DBMS.

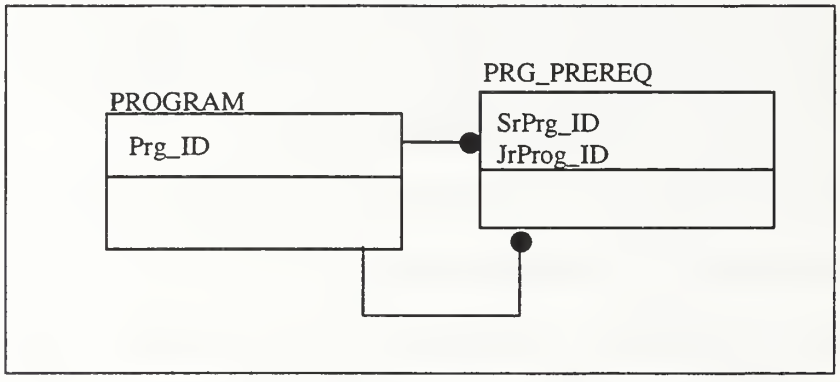

Figure 4-4. Resolved Recursive Relationship

Additionally, the issue of normalization must be considered, regardless of the target DBMS. If normalization was accomplished at the entity level during the development of the conceptual data model, than a normalized schema will be generated automatically. However, if normalization did not occur at the entity level, than all generated tables must be checked for normalization after schema generation. Necessary

modifications to the tables can occur at that time, and a corresponding relational model can be reverse engineered to reflect the new structure of the tables.

Other issues related to schema generation are DBMS dependent. Several issues regarding the use of an Oracle DBMS are discussed in this section. These issues include: 1) Enforcing referential integrity; 2) avoiding the Oracle mutating table error; and 3) importing data definitions to Oracle.

1. Enforcing Referential Integrity

Once the IDEF1X data model was created, the ER*win* schema generation function was used to generate the Oracle schema. MCI established several business rules that were beyond the enforcement capabilities normally associated with Oracle referential integrity rules. For example, for the referential integrity type, "on parent update", an MCI business rule requires the ability to cascade an update on the student's social security number into all associated tables. Since Oracle strictly enforces the restriction of updating parent table primary keys when foreign keys are defined using SQL, it was decided to enforce referential integrity and define relationships through the use of server side triggers instead of SQL. These triggers are discussed in more detail in Section B.

In order to accommodate this business rule allowing for social security numbers to be altered, the ER*win* trigger template feature was used to enforce the referential integrity rule, "on parent update cascade". This allows the user to alter a student's social security number and cascades that change through all associated tables. This rule can only be enforced through the generation of server side triggers, and the simultaneous disabling of the ER*win* feature normally used to generate SQL code which defines foreign key relationships.

2. Avoiding the Oracle Mutating Table Error

A mutating table error occurs when a trigger or a user-defined PL/SQL procedure or function attempts to query or modify a table that is in the middle of being modified by the statement that fired the trigger. It is a very common and frustrating error that frequently occurs when a business rule requires that the table or row being modified comply with certain conditions. In this case, a query is conducted on the table to check these conditions at the same time a write to the table is being performed.

In order to avoid the Oracle mutating table error (Oracle server error number ORA-04091 -- table is mutating, trigger/function may not see it), a series of ER*win* generated built in packages is used to generate a series of server side triggers, functions, and procedures. These modules use temporary variables to store new table values. The temporary variables are used as comparison operators with the values returned from the query.

If the conditions required by the associated business rule are met, the values are then written from the temporary variables to the table. If not, an appropriate alert dialogue box is displayed to the user. The steps necessary to employ these built in ER*win* packages are defined in Appendix Q.

3. Specifying Data Definitions in ER*win*

ER*win* allows for all entities, attributes, and relationships to be defined. These definitions can be imported into the Oracle schema as comments. It is possible to read these comments directly from Oracle; however, it is recommended that ER*win* be used to develop all data dictionary reports. ER*win* permits the development of custom dictionary reports, and the generation of these reports in any of several popular word processing or

spread sheet applications, such as Microsoft Word and Microsoft Excel. All entities, attributes, and relationships present in the IDEF1X model developed for this project have been explicitly defined in ER*win*.

B. SERVER SIDE TRIGGERS

This section describes the use of server side triggers to enforce MCI's business rules. A series of triggers were developed to demonstrate the benefits of using server side triggers as a means of enforcing business rules.

1. Using Triggers to Enforce Referential Integrity

As indicated earlier, all triggers enforcing referential integrity were generated automatically by ER*win*. This was done in part to enforce the cascading updates of parent primary keys. In order to avoid the mutating table error, a series of procedures and functions were utilized which permit the temporary storage of data to be updated in PL/SQL tables.

These are not relational tables in the traditional sense. Rather, they are records of arrays. Each PL/SQL table is actually an array consisting of two columns: The data column and an identifier column, the value of which is generated automatically by the PL/SQL block. Several of these arrays are combined in a function or procedure to simulate an Oracle table. Values to be added to the database are stored in these simulated tables and checked by the appropriate procedures for compliance with MCI's business rules. If the values are considered valid, the appropriate insert, update, or deletion is permitted.

In this manner, all referential integrity rules can be enforced via server side triggers. If the business rules concerning referential integrity change in the future, it is

important that these changes be carefully executed. In order to properly modify the triggers enforcing integrity, the following steps must be followed.

First, the associated triggers and their corresponding procedures and functions must be deleted from the Oracle schema. This step is most easily accomplished by using the appropriate Oracle GUI database manager interface, such as Navigator for Oracle Personal 7 or Enterprise Manager for Oracle Server.

Once the triggers and their associated procedures and functions are deleted, the ER*win* trigger template editor must be modified to reflect the new referential integrity rule. For example, it might be changed from parent update cascade to parent update restrict, which would thereby prohibit the modification of primary keys.

After the appropriate change is made in ER*win*, a new schema must be generated. This will cause ER*win* to automatically generate the appropriate triggers and corresponding procedures and functions. If these triggers are generated without first deleting the triggers, procedures, and functions associated with the old business rule, data integrity could be violated. When the new schema is generated, ER*win* ***does not delete the old PL/SQL blocks from the schema***. Complicating matters further, in cases where the new blocks have identical names to the old blocks, the old blocks are not overwritten. In order to insure that the proper business rules are enforced, it is important to manually delete outdated triggers, procedures, and functions prior to generating the updated schema. This should also be done whenever relationships between entities are added, modified, or deleted.

2. Using Triggers to Enforce Various Business Rules

In addition to those triggers generated automatically by ER*win* to enforce referential integrity, several triggers, and their associated packages containing PL/SQL stored procedures and functions, were manually implemented in order to demonstrate the suitability of server side triggers for the enforcement of various business rules. In several cases, these triggers utilize PL/SQL tables in the manner described above to avoid mutating table errors.

Triggers were written to demonstrate the use of server side triggers, procedures, and functions to enforce various business rules. They cover 4 major areas: Transaction records, program enrollment and automatic enrollment in associated courses, updating student records based on MCTF downloads, and the enforcement of eligibility requirements based on grade and service component. These triggers provide the following functionality:

- The appropriate transaction code is automatically entered in the student course enrollment transaction log (i.e., STUD_CRS_TRAN_X table) when a student is enrolled in a course (i.e., an instance is inserted into the STUD_CRS_X table).

- The appropriate transaction code is automatically entered in the student program enrollment transaction log (i.e., STUD_PRG_TRAN_X table) when a student is enrolled in a program (i.e., an instance is inserted into the STUD_PRG_X table).

- When a new instance is inserted into the STUD_PRG_X table, course enrollments are inserted or updated in the STUD_CRS_X table for all courses associated with the program. That is, when students enroll in a program, they are automatically enrolled in all associated courses.

- Whenever the Marine Corps Total Force table (MCTF_PERS) is overwritten or updated, all matching records in the STUDENT table are checked and modified as required to match the MCTF_PERS table. Thus, if a Marine who

is a student changes his name, that change will be automatically made in the STUDENT table when the MCTF_PERS table is loaded from the total force database.

- When an attempt is made to enroll a student in a program, the student's current pay grade and service component is checked. If the student is not eligible for enrollment based on pay grade and service component, the insert into the STUD_PRG_X table is prohibited, and the appropriate error message is displayed.

The above examples clearly illustrate the usefulness of utilizing server side triggers, functions, and procedures to enforce business rules. By maintaining the code used to enforce business rules on the server, changes to business rules are easily enforced, since they are implemented in a single central location. The alternative of maintaining triggers on the application side requires modifying the rules on every copy of the application that uses the rule when a change is needed. The PL/SQL code associated with the manually coded triggers described above is included in Appendix R.

3. Dangers Associated With Using Stored Procedures and Triggers

The use of built-in procedural extensions, including stored procedures and triggers, to enforce referential integrity and other business rules is considered to be extremely non-standard [Ref. 5]. It is non-standard because database servers traditionally manipulate data and enforce business rules without relying on procedural language constructs. In a sense, the use of procedural constructs violates a cardinal rule of traditional client/server systems: Instead of keeping data separate from procedural code, the code is now being stored in the database in the form of stored procedures.

Stored procedures are a combination of SQL statements and procedural logic (e.g., PL/SQL code) that is compiled, verified, and stored in the target database. Oracle treats

stored procedures like any other database object, and access to the procedures is controlled by the DBMS. Triggers are special user-defined actions, in the form of stored procedures, that are invoked automatically by the DBMS when certain conditions are met. As stated earlier, in order to allow for the update of parent keys and the enforcement of various other business rules, stored procedures and triggers were used to enforce referential integrity and business rules in this prototype relational schema. This presents a variety of concerns.

Using triggers to enforce referential integrity is extremely non-standard, error-prone, and hazardous to implement and maintain [Ref. 5]. The server is unaware that the user is relying on triggers to enforce referential integrity; therefore, the server is unable to optimize the enforcement of the referential integrity rule. For example, if a batch process adds 100 students to the database as part of a group enrollment in a particular course, the trigger prohibiting the enrollment of a single student in the same course more than once will fire 100 times. If, on the other hand, the server resident referential integrity checker was used instead of the trigger, this check could be optimized by the server and performed in the most efficient manner.

Depending on triggers to enforce referential integrity is also dangerous because the use of the triggers is difficult to document, and programming errors can lead to unexpected results [Ref 5]. Declarative integrity based on server resident integrity checkers is clearly documented and dependable. Additionally, most DBMSs limit the number of triggers supported per table, which may proscribe the enforcement of all referential constraints and business rules defined by the user. This final limitation is not considered to greatly affect Oracle 7 servers, as they support up to 12 triggers per table.

C. SCHEMA MAINTENANCE IN ER*WIN*

ER*win* should be used to facilitate the maintenance of the MCI database. All changes to entities, attributes, and relationships should be made in ER*win*. Once these changes are made, a new Oracle schema can be generated. This approach will keep the data model and schema synchronized all the time, and greatly facilitate the maintenance of the database. Additionally, it is recommended that all changes to data definitions be made in ER*win*, in order to maintain accurate documentation of all data definitions.

If changes are inadvertently made directly to the Oracle schema, The conceptual data model can be reverse engineered using ER*win*. ER*win* can read the data dictionary from many different types of DBMS servers, and can use this information to automatically generate an IDEF1X model and accompanying dictionary. This model can be compared programatically by ER*win* with the file containing the original data model.

This allows for the user to verify the accuracy of the current database, as well as identify changes which have occurred since initial design. Another benefit yielded by such an approach is the ability to rapidly expand and contract the conceptual data model and accompanying database as user requirements change. This makes the data model a living document, with current documentation throughout its life cycle. Finally, the proposed relational data model for MCI requires constant verification from the database users to ensure its validity and accuracy.

VI. DATA MIGRATION ISSUES

This Chapter discusses the issues involved in migration of existing data. It discusses: 1) format and location of existing data; and 2) a strategy for migration.

A. FORMAT AND LOCATION OF EXISTING DATA

MCI's current automated data system runs on a Hewlett-Packard 3000 minicomputer running the MPE/iX operating system. MCIAIS is written in the HP proprietary language "Transact" and accesses a TurboIMAGE hierarchical database. As such, it does not have underlying data or process models. Current data resides in a "closed" non-relational database. Additionally, data locations are poorly documented. Much of the documentation provided by MCI to this project team proved to be outdated and erroneous.

In order to accurately determine the format and location of existing data, a copy of the IDEF1X model and associated data dictionary was provided to Mr. Rudd in January 1997. Working from the data dictionary, Mr. Rudd documented the source location of existing data in the TurboIMAGE hierarchical database. Comments were provided which amplified the source location information. This attribute availability report is provided as Appendix S.

Several minor modifications occurred in the data model between January 1997 and the time of this report. Some attributes and entities exist in the current data model that were not present when Mr. Rudd prepared the attribute availability report. For this reason, availability of attributes included in the current model requires some additional verification prior to data migration.

B. DATA MIGRATION STRATEGY

Data migration follows three basic steps: 1) Identification of attributes currently available; 2) Mapping and migration of currently available attributes to the newly developed database; and 3) automatic input of attributes not currently available at some future time (once they become available through automation and redesign of other MCI systems and departments), or manual input of attributes not currently available and not likely to be made available through the automation of other MCI systems or departments.

Identification of currently available attributes began with the development of the attribute availability report by Mr. Rudd. Once this report is verified by MCI, mapping and migration of these attributes to the new database can begin. This verification should occur just prior to migration, as this allows for the incorporation of all recent changes to existing data structures into the migration plan.

In order for migration of existing data to be successful, the following requirements must be met: 1) All mandatory column values (e.g., primary keys, not null attributes) for all tables to be populated must be available; and 2) data being migrated must be of the correct data type and field length, as reflected in the current data dictionary.

C. DATA MIGRATION CONSTRAINTS

In many cases, mandatory column values are not available in the existing data stores. In these instances, it is required that the data model be *temporarily modified* in a manner that makes these mandatory attributes nullable, for the purpose of migration. For example, many of the attributes associated with the course entity are not available. Attributes such as EdLev, CrsCreditHrs, CrsDesc, and CrsDesignedFor are available in the current course catalogue, but not in the TurboIMAGE database. Thus, these attributes

should be changed from "not null" to "null" in the schema prior to migration of current course data from the MCIDB database. In some cases, mandatory columns exist as foreign keys. In these instances, relationships must be modified such that their minimum cardinality is zero (optional) .

D. DATA MIGRATION IMPLEMENTATION

Once all mandatory column values are available, and columns not available are made nullable, data migration can begin. Scripts must be written which read the available data from the legacy database and populate the relational tables. In many instances, this data will require type and field length conversion.

Wherever practical, the data model is designed such that data types and field lengths in the new relational schema correspond to existing data types and field lengths. Discrepancies are annotated in the comments column of Appendix S. Where feasible, further modification of the data model should be made so that data types and field lengths in the relational table match the corresponding TurboIMAGE data types. These data types and field lengths can then be converted to more appropriate types through the use of PL/SQL scripts, if required.

Data elements not currently available must be entered manually, either prior to the data migration of available data or at some future time. Data elements for all domain tables, with the single exception of the MCTF_PERS table, should be entered manually prior to data migration. These tables contain minimal data, and serve as definition sources for domains associated with attributes in most of the main entities. The MCTF_PERS table can be populated prior to data migration from the existing Marine Corps Total Force System, by utilizing a loading utility such as SQL Loader.

Several data elements will be available for automated migration (i.e., manual entry will not be required), after redesign of the LOGAIS database. For instance, on hand quantities of courses, programs, job aids, components, and training NCO material should not be accomplished manually. These attributes should not be entered prior to redesign of the LOGAIS inventory system. After LOGAIS is redesigned, these attributes will be available for automatic entry into the MCIAIS system.

The migration of the current MCIAIS database to a new database stresses the importance that future development of a relational database to support LOGAIS should be done with proper consideration of the MCIAIS model. Data types, field lengths, and corresponding data definitions which will be shared by the two systems must be identical.

Manual input of some data elements cannot be avoided. In some cases, MCI should elect to forego entry of past data and begin populating tables with current transactions. For example, it may be of little added value to populate the STUD_CRS_TRAN_X table with all past transactions, which are currently stored in seven byte arrays. In this case, it would be more realistic to maintain access to the existing arrays of past transactions, and begin populating the relational table with transactions occurring after implementation of the relational database.

The majority of manual inputs required are associated with attributes created, updated, and deleted by MCI activities outside of SSD. Most of this data is not currently available in electronic form, and its input should be avoided until relational tables to support these activities exist in the MCI database. For example, the cataloging and entry of components and component descriptions must be postponed until the occupational specialty department and professional military education department are fully automated,

as the creation and modification of these attributes are the responsibility of these departments.

Similarly, population of the MULT_CHOICE and EXAM tables will require either a massive manual data entry effort by SSD or MIS personnel, or the automation of the occupational specialty department, as that department is responsible for the development of multiple choice exams.

Thus, the data migration effort associated with the population of the relational schema developed for this project requires:

1. Verification of the attribute availability report (Appendix S).

2. Modification of the tables such that all attributes not currently available are nullable.

3. Manual entry of data in domain tables.

4. Population of the MCTF_PERS and RUC tables via download from the Marine Corps Total Force System and the use of a SQL loading utility.

5. Writing scripts using an appropriate language to map currently available data into the newly developed relational tables. In many instances, conversion of data types and field lengths must occur in this process.

6. Future migration of data obtained from relational tables developed to support LOGAIS and other MCI departments, once they are redesigned and fully automated.

7. Manual entry of data not currently available.

8. Manual entry of data available but not able to be exported from existing files to relational tables.

It is strongly recommended that a migration gateway such as that marketed by StarVision, LLC. be used to accomplish the migration of data from TurboIMAGE to Oracle.

VII. CONCLUSIONS AND RECOMMENDATIONS

This chapter presents the conclusions and recommendations of this research. First, it addresses the research objectives and research questions that were stated in Chapter I of this thesis. Next, it summarizes the various practical strategies for developing and maintaining relational databases. It then presents recommendations for implementation of relational systems, and summarizes the technical obstacles to implementation. Additionally, applicability of this research to other systems at MCI (e.g., LOGAIS) is briefly discussed. Finally, future research requirements and opportunities for further research are suggested.

A. ACHIEVEMENT OF RESEARCH OBJECTIVES AND QUESTIONS

This research is the culmination of a year long project commissioned by MCI to develop the architecture and supporting migration plan to transition from a closed, non-relational system to an open, client/server based relational database management system (DBMS). The objectives of this research project were achieved: A detailed analysis of data requirements at MCI was performed in conjunction with a review of the existing non-relational database. Development of a conceptual data model, using the IDEF1X technique, and subsequent generation of a relational database schema, was accomplished. The proposed relational database will increase efficiency, reduce costs, and support future system evolution. The developed data migration plan and comprehensive data dictionary will facilitate the migration effort at MCI. Additionally, the strategies developed for data and database maintenance will assist MCI in future system evolution. In the course of this project, the research questions posed in Chapter I of this thesis were answered. These questions and the answers provided by this research are outlined below.

1. Primary Research Questions and Answers

- **Can a data model be developed in IDEF1X to support the current and future needs of the student services department (SSD) at MCI?**

The effort addressed by this thesis resulted in the development of an IDEF1X data model capable of supporting the current and future needs of the student services department at MCI. This model adheres to the DoD standard for data modeling, and can be expanded or contracted to meet the needs of MCI into the foreseeable future.

- **Can existing data in the Marine Corps Institute's Automated Information System (MCIAIS) be successfully migrated from the current legacy system to an open client/server system based on a relational database?**

Whether MCI can execute a successful migration remains to be seen. This research proposes a plan for the successful migration of the existing MCIAIS into a client/server, open system. Additionally, the recent development of commercial migration gateways will increase the chances of a successful migration. If MCI adheres to the migration plan recommended in this thesis, a successful migration is technically possible. However, political considerations, beyond the scope of this research, could adversely impact the migration effort.

2. Subsidiary Research Questions and Answers

- **How successful are available CASE tools in supporting data modeling using the IDEF1X methodology?**

During the course of this research, ER*win* has proven to be a capable CASE tool, which adequately supports the use of the IDEF1X methodology and enhances the ability to maintain current documentation on both the data model and the associated relational schema.

- **How effectively do data modeling tools support other stages of the system development life cycle by interfacing with process modeling CASE tools?**

The ability of ERw*in* to support other stages of the system development life cycle by interfacing with process modeling CASE tools was considered to be marginal. For example, the interface between ERw*in* and its sister process modeling tool, BPw*in*, is poor and inadequately documented. The tools do not share a common data repository or common dictionary, and depend on import and export functions to provide the necessary interface. As a result, importing entities and attributes from ER*win* to BP*win* results in importing of names only; data definitions are not imported. This shortcoming left the process re-engineering team frustrated at several critical junctures during model development.

- **How effectively does the target DBMS support the need for a robust client/server based relational DBMS?**

Oracle 7 effectively supports the need for a robust client/server based relational DBMS. PL/SQL is a valuable tool for enforcing business rules on the server, and allows for the enforcement of referential integrity rules normally disallowed by Oracle, such as the cascade update of parent keys.

B. SUMMARY OF DEVELOPMENT AND MAINTENANCE STRATEGIES

This thesis developed and advocates the use of various practical strategies for developing and maintaining relational databases. Development should follow the four stages described above, including: 1) listing candidate data entities, 2) defining the entities and their associated attributes and relationships, 3) developing the data model, and 4) relating the entities to the business processes to reveal candidate applications. IDEF1X methodology should be employed for the development of all DoD relational

databases, as it is the standard methodology, and is well defined and clearly documented in FIPS PUB 184.

Maintenance requirements are eased through the use of a suitable CASE tool, such as ER*win*. CASE tools facilitate synchronization of the data model with the relational schema and automate most documentation requirements. Additionally, they facilitate the publication and maintenance of a user-friendly data dictionary.

C. IMPLEMENTATION RECOMMENDATIONS

MCI should implement the relational database proposed in this thesis in an open, client/server environment. The benefits enjoyed by operating in such an environment, when coupled with the advantages that a relational database enjoys over the current file-processing based system, will greatly enhance the functionality of MCI and increase its ability to fulfill its mission of developing, publishing, distributing, and administering distance training and education to Marines.

Migration of data from the current legacy system to the future open, client/server based relational DBMS should follow the three steps outline in Chapter VI:

1. Identification of attributes currently available.

2. Mapping and migration of currently available attributes to the newly developed database.

3. Automatic input of attributes not currently available at some future time (once they become available through automation and redesign of other MCI systems and departments) , or manual input of attributes not currently available and not likely to be made available through the automation of other MCI systems or departments.

Success of this migration requires that all mandatory column values (e.g., primary keys, not null attributes) for all tables to be populated must be available and that data being

migrated must be of the correct data type and field length, as reflected in the current data dictionary.

D. ANTICIPATED OBSTACLES

Non-trivial organizational issues such as politics, cultural bias, fiscal limitations, and top-level leadership support must be considered. IS managers must be able to address these challenges effectively, otherwise the technical issues discussed in this work will have little impact on the success of future system deployment. These issues, however, are beyond the scope of this research.

E. APPLICATION FOR OTHER SYSTEMS

This work advocates a specific methodology for the development of a target relational database. This methodology is independent of the implementation data model and is also independent of any CASE tool used in model development and future system maintenance. As such, it is applicable to all migration efforts from non-relational databases to relational databases installed on an open, client/server architecture.

At MCI, the scope of this research effort should be expanded to include all aspects of the enterprise-wide data model, including migration of the current LOGAIS database to a relational database. It is anticipated that a single relational database accessed via an open client/server architecture can meet all of the enterprise-wide data requirements at MCI. Additionally, only with the migration of the LOGAIS data to a relational system and the implementation of a fully automated inventory system will the full potential of the SSD system be realized. Adequate customer service support cannot be accomplished without on-line access to current inventory and invoice data.

F. FUTURE RESEARCH REQUIREMENTS

The Oracle 7 hybrid architecture is relatively new and overcomes many of the shortcomings associated with the process-per-client architecture upon which previous releases of Oracle were based. This improved architecture allows for reusable shared server processes and provides a protected environment for each user task, without requiring each client to have a permanently assigned process address space.

Whether the effects of queue latencies from which the hybrid architecture suffers will adversely impact processing at MCI remains to be seen, as processing demands on the prototype where not of a sufficient quantity or duration to test the issue of latent queues. Additionally, the impact of these queues on the TP monitor's scheduling algorithms remains untested. As stated earlier, performance with this architecture can be expected to range from twenty percent improvement to a twenty percent degradation in comparison with previous relational architectures. There is no doubt, however, that this architecture will be an improvement over the current file-processing system utilized by MCI.

G. CONCLUSIONS

The process of answering the research questions posed at the beginning of this thesis reveals a common sense, logical strategy for accomplishing the design of a relational DBMS, the migration of existing data from a non-relational database to the new system, and the subsequent maintenance and evolution of the new system. These strategies should prove valuable for IS managers who are struggling to develop relational databases in support of their organization and attempting to migrate from existing file-processing systems to those newly implemented relational systems.

Failure to implement client/server based relational systems can be catastrophic. Legacy system maintenance can lead to a death spiral of inadequate systems. The expense of maintaining legacy systems based on closed, non-relational databases prohibits the allocation of resources for new system development. Inadequate development feeds the inadequate current system and spawns more legacy systems. Organizations are caught in a downward spiral of supporting legacy systems, ignoring future needs, and inadequately planning and budgeting for expanding operational commitments.

To escape the grip of maintaining legacy, non-relational databases, the costs associated with maintaining these systems must be leveraged [Ref. 3]. Life-cycle costs must be considered, and the organization's long term objectives must be weighed against the organization's short term needs. Maintaining a clear focus on long term objectives and creating a vision which supports these objectives will allow the development of a system which is capable of evolving with changing business requirements. Relational databases are consistent with this need to develop systems which are highly flexible, interoperable, and have the ability to evolve over time.

In order for DoD organizations to effectively deploy, maintain, and operate relational databases, the basic concepts of relational systems must be fully understood by the personnel responsible for system design and operation. As there are a variety of modeling techniques available, as well as numerous CASE tools which support these techniques, managers face the dilemma of which tools and strategies to select when dealing with the migration from legacy databases to open, relational systems. This research purports that the use of the strategies described above, coupled with the IDEF1X

methodology and a suitable CASE tool which supports synchronization of data models with relational schemas, will allow for the successful migration to and maintenance of open, client/server relational databases.

LIST OF REFERENCES

1. Kroenke, David M., *Database Processing*, Prentice Hall, 1995.

2. Brodie, Michael L. and Stonebraker Michael, *Migrating Legacy Systems*, Morgan Kaufmann Publishers, Inc., 1995.

3. Cameron, Robert A. and Carrick, Kenneth G., *Reengineering DOD Through Enterprise-wide Migration to Open Systems,* Thesis, Naval Postgraduate School, Monterey, California, September, 1996.

4. Marine Corps Institute, *Abbreviated System Decision Paper for The Redesign of the Marine Corps Institute's Automated Information System (MCIAIS)*, unpublished report, MCI, Washington D.C., 1996.

5. Orfali, Robert, Harkley, Dan, and Edwards, Jeri, *Essential Client/Server Survival Guide,* John Wiley and Sons, Inc., 1994.

6. Batini, Carlo, Ceri, Stefano, and Navathe, Shamkant B., *Conceptual Database Design: An Entity-Relationship Approach*, The Benjamin/Cummings Publishing Company, Inc., 1992.

7. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *CACM* 13, No. 6, June 1970.

8. Date, C. J., *Relational Database: Selected Writings*, Addison-Wesley Publishing Company, Inc., 1986.

9. Codd, E. F., *The Relational Model For Database Management, Version 2*, Addison-Wesley Publishing Company, Inc., 1990.

10. Kamel, Magdi N., and McCaffrey, Martin J., *Tomahawk Engineering 2000 Project: Database Analysis Study Final Report*, unpublished report, Naval Postgraduate School, Monterey, California, December, 1995.

11. Federal Information Processing Standards Publication 184, *Integration Definition for Information Modeling (IDEF1X)*, Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Systems Laboratory, Gaithersburg, Maryland, December 21, 1993.

12. *Logic Works* ER*win, Getting Started,* Logic Works, Inc. 1996.

13. Feuerstein, Steven, *ORACLE PL/SQL Programming*, O'Reilly and Associates, Inc., 1995.

14. Kamel, Magdi N., et al., *Analysis, Design, and Prototype Implementation of a Contemporary Information System for the Marine Corps Institute: Preliminary Report*, Naval Postgraduate School Technical Report, Naval Postgraduate School, Monterey, California, February, 1997.

15. Kamel, Magdi N., et al., *Analysis, Design, and Prototype Implementation of a Contemporary Information System for the Marine Corps Institute: Final Report*, Naval Postgraduate School Technical Report, Naval Postgraduate School, Monterey, California, June, 1997.

APPENDIX A. ACRONYMS AND TERMS

ANSI	American National Standards Institute
ANSREF	Exam Answer and Reference Database
API	Application Programming Interface
ARCHIV	Archived Student Records Database
AWS	Amphibious Warfare School
CASE	Computer Aided Software Engineering
CPU	Central Processing Unit
CRUD	Create Read Update or Delete
DBMS	Database Management System
DDL	Data Definition Language
DoD	Department of Defense
FIPS	Federal Information Processing Standards
FIPS PUB 184	Federal Information Processing Standards Publication 184
GUI	Graphical User Interface
HP	Hewlett Packard
IDAPI	Integrated Database Application Programming Interface
IDEF1X	Integration Definition for Information Modeling
ISO	International Standards Organization
JPME	Joint Professional Military Education
LOG	Logistics Department
LOGAIS	Logistics Automated Information System
MCI	Marine Corps Institute
MCIASS	Marine Corps Institute Automated Support System
MCIAIS	Marine Corps Institute's Automated Information System
MCIDB	Student Course and Information Records Database
MCTF	Marine Corps Total Force
MIS	Management of Information Systems (Department)
MMSDB	Active Duty Marines Database
MSEXAM	Exam Stock Status Database
NIST	National Institute of Standards and Technology
ODBC	Open Database Connectivity
OPS	Education and Operations Department
OS	Operating System
OSD	Occupational Specialty Department
PL/SQL	Procedural Language Extensions to SQL
PME	Professional Military Education
PMED	Professional Military Education Department
RMPDB	Reserve Marines Database
RUC/MCC	Reporting Unit Code/Major Command Code
SALEDB	Statistical Analysis of Lessons and Exams Database
SQL	Structured Query Language

SSD	Student Services Department
SSN	Social Security Number
TP	Transaction Processing
USMC	United States Marine Corps

APPENDIX B. MCI ENTERPRISE LEVEL CANDIDATE DATA SUBJECTS

Data Subject Name
ADVERTISMENT
COMPONENTS
COPY_MATERIAL
COURSE
COURSE_COPY_MATERIAL
COURSE_DEVELOPERS
CUSTOMER
EVENTS
EXAM
FINANCIAL
INVENTORY
IS_EQUIPMENT_INVENTORY
ISSUE_COMPLAINT
JOB_AID
JOB_AID_COPY_MATERIAL
MCI_PERSONNEL
MCTF_PERS
ORDER
PROGRAM
PROGRAM_COPY_MATERIAL
PROGRAM_DEVELOPERS
PURCHASE
SSD_PERSONNEL
STUDENT
TRAINING
WAREHOUSE

APPENDIX C. MCI ENTERPRISE LEVEL DATA SUBJECT DEFINITIONS

Data Subject Name	Data Subject Definition
ADVERTISMENT	Contains information on advertising projects. Advertising projects are related to courses and programs
COMPONENTS	Table of all items stocked by MCI. This includes items included in courses and programs(lessons, course material, dictionaries, etc.) as well as items included in job-aids (FAC manuals, etc.)
COPY_MATERIAL	Copy Material is any text or graphic material which must be written or produced by a developer or writer. It is stored electronically as binary data for future reproduction or distribution as either digital or paper material. It includes such things as exams, course material, program books and material, etc. Due to the size of some material, it may be stored on some backup storage media. If this is the case, this record will only contain a pointer to indicate the location of the material. Smaller material may be stored directly in the database.
COURSE	All active, closed, and future courses as entered by the course developer.
COURSE_COPY_MATERIAL	copy material which is associated with a course
COURSE_DEVELOPERS	MCI Course developers and writers
CUSTOMER	Anyone who contacts MCI with a question. Contact may be by telephone, e-mail, in person, regular mail, etc.
EVENTS	Parade and VIP events staffed or supported by MCI
EXAM	An instance of a specific exam as generated for a specific course
FINANCIAL	Contains budgeting information. Certain line items may be associated with the funding designated for the development of courses and programs
INVENTORY	An inventory record for a component
IS_EQUIPMENT_INVENTORY	Contains the inventory records of all IS Equipment
ISSUE_COMPLAINT	A customer generates an issue or a complaint
JOB_AID	Table of all job aids stocked by MCI.
JOB_AID_COPY_MATERIAL	Copy material which is associated with a job aid
MCI_PERSONNEL	All personnel employed at MCI
MCTF_PERS	Table of all MCTF personnel as downloaded from the Total Force database
ORDER	An order is any request by a customer for components, job aids, courses, or programs other than an actual enrollment in a course or program
PROGRAM	All active, closed, and future programs as entered by the program developer.
PROGRAM_COPY_MATERIAL	Copy material which is associated with a program
PROGRAM_DEVELOPERS	MCI Program developers and Writers
PURCHASE	All course components are purchased. An order is placed by a MCI clerk and filled by a supplier.
SSD_PERSONNEL	Personnel working in the Student Services Department
STUDENT	All Students contained in the MCI database
TRAINING	Training programs for MCI course writers and programmers are described here.
WAREHOUSE	A warehouse is not just the main warehouse or warehouses belonging to logistics. As any department may potentially store material, there may be one or more warehouses associated with each department

APPENDIX D. MCI ENTERPRISE LEVEL PRIMARY KEYS

Entity Name	Entity Type	Primary Key	Foreign Keys
ADVERTISMENT	Independent	Project_ID	
COMPONENTS	Independent	Comp_ID	
COPY_MATERIAL	Dependent	Comp_ID Material_ID	Component_ID References: COMPONENTS
COURSE	Independent	Crs_ID	
COURSE_COPY_MATERIAL	Dependent	Comp_ID Material_ID	Material_ID, Component_ID References: COPY_MATERIAL
COURSE_DEVELOPERS	Dependent	SSN_ID	SSN_ID References: MCI_PERSONNEL
CUSTOMER	Independent	CustSSN_ID	
EVENTS	Independent	Event_ID	
EXAM	Dependent	Crs_ID Exam_ID	Course_ID References: COURSE
FINANCIAL	Independent	LineItem Year	Course_ID References: COURSE Program_ID References: PROGRAM
INVENTORY	Dependent	Comp_ID Warehouse_ID	Component_ID COMPONENTS Warehouse_ID References: WAREHOUSE
IS_EQUIPMENT_INVENTORY	Independent	Serial_No	
ISSUE_COMPLAINT	Dependent	CustSSN_ID IssueNumber	CustomerSSN_ID References: CUSTOMER
JOB_AID	Independent	JobAid_ID	
JOB_AID_COPY_MATERIAL	Dependent	Comp_ID Material_ID	Material_ID, Component_ID References: COPY_MATERIAL
MCI_PERSONNEL	Independent	SSN_ID	MCTFSSN_ID References: MCTF_PERS

Entity Name	Entity Type	Primary Key	Foreign Keys
MCTF_PERS	Independent	MCTFSSN	
ORDER	Dependent	CustSSN_ID InvoiceNumber	CustomerSSN_ID References: CUSTOMER
PROGRAM	Independent	Program_ID	
PROGRAM_COPY_MATERIAL	Dependent	Comp_ID Material_ID	Material_ID, Component_ID References: COPY_MATERIAL
PROGRAM_DEVELOPERS	Dependent	SSN_ID	SSN_ID References: MCI_PERSONNEL
PURCHASE	Independent	Order_ID	
SSD_PERSONNEL	Dependent	SSN_ID	SSN_ID References: MCI_PERSONNEL
STUDENT	Independent	StudSSN_ID	MCTFSSN_ID References: MCTF_PERS
TRAINING	Independent	TrainingProg_ID	
WAREHOUSE	Independent	Warehouse_ID	

APPENDIX E. MCI ENTERPRISE LEVEL CANDIDATE ATTRIBUTES

Base Name
Addr1
Addr2
BldgNumber
BudgetAmount
City
Comments
CommercialNo
Component
ComponentDescription
ComponentName
Component_ID
CopyMaterial
CourseAbbreviation
CourseNumber
Course_ID
CreditHours
Custodian
CustomerSSN_ID
DCTB
DSNNumber
Date
DateClosed
DateOpened
Department
Description
DesignedFor
ECC
ECC
EDD
EventCoordinator
EventDate
EventDesc
EventLocation
EventName
EventTime
Event_ID
Exam_ID
FirstName
Grade
InvoiceNumber
IssueNumber
JobAidDescription
JobAidName
JobAid_ID
JobDescription
LastName
LineItem
Location

Base Name
MCTFSSN_ID
MOS
Manager
Material_ID
MiddleInitial
Nomenclature
NumberOfQuestions
OccFieldSpec
OnHand
OrdStatDate
OrderDate
Order_ID
PDD
PassingScore
Phone
PlatoonCode
Pointer
ProgramAbbreviation
Program_ID
Project_ID
Quantity
RDD
Rank
ReserveCredits
SSN_ID
SalesRep
SchoolCode
Section
SelToGrade
Serial_No
State
StudyHours
Supplier
Title
TrainingProg_ID
Warehouse_ID
Year
ZipCode

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
Addr1	CUSTOMER	First line of customer's address	VARCHAR2(46)	NOT NULL
Addr1	MCTF_PERS	First address line of the Home Address for a Marine contained in the MCTFS database. Used as the mailing Address for Reservists	VARCHAR2(46)	NOT NULL
Addr1	STUDENT	First line of student's mailing address	VARCHAR2(46)	NULL
Addr2	CUSTOMER	Second line of customer's address	VARCHAR2(46)	NULL
Addr2	MCTF_PERS	Second Address line of the Home Address for a Marine contained in the MCTFS database. Used as the mailing Address for Reservists	CHAR(46)	NULL
Addr2	STUDENT	Second line of student's mailing address	VARCHAR2(46)	NULL
BldgNumber	WAREHOUSE	Building number of the warehouse	VARCHAR2(3)	NOT NULL
BudgetAmount	FINANCIAL	Amount budgeted	NUMBER	NOT NULL
City	CUSTOMER	City of customer's address	VARCHAR2(34)	NOT NULL
City	MCTF_PERS	City of the Home Address for a Marine contained in the MCTFS database. Used as the mailing Address for Reservists	VARCHAR2(34)	NULL
City	STUDENT	City of student's mailing	VARCHAR2(34)	NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		address		
Comments	ISSUE_COMPLAINT	comments regarding the complaint or issue	VARCHAR2(2000)	NOT NULL
Comments	PURCHASE	Any comments associated with the order. Add for clarity.	VARCHAR2(2000)	NULL
CommercialNo	CUSTOMER	Commercial telephone number where customer can be reached	CHAR(10)	NULL
Component	MCTF_PERS	Component of a Marine contained in the MCTFS database.	CHAR(2)	NOT NULL
ComponentDescription	COMPONENTS	Narrative description of each item	VARCHAR2(60)	NOT NULL
ComponentName	COMPONENTS	Name associated with each component as provided by the LOGAIS system.	VARCHAR2(30)	NOT NULL
Component_ID	COMPONENTS	Unique ID associated with every component. Ideally, these values will be provided and updated via interface with LOGAIS inventory software.	VARCHAR2(10)	NOT NULL
Component_ID	COPY_MATERIAL	Unique ID associated with every component. Ideally, these values will be provided and updated via interface with LOGAIS inventory software.	VARCHAR2(10)	NOT NULL
Component_ID	COURSE_COPY_MATERIAL	Unique ID associated	VARCHAR2(10)	NOT

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		with every component. Ideally, these values will be provided and updated via interface with LOGAIS inventory software.		NULL
Component_ID	INVENTORY	Unique ID associated with every component. Ideally, these values will be provided and updated via interface with LOGAIS inventory software.	VARCHAR2(10)	NOT NULL
Component_ID	JOB_AID_COPY_MATERIAL	Unique ID associated with every component. Ideally, these values will be provided and updated via interface with LOGAIS inventory software.	VARCHAR2(10)	NOT NULL
Component_ID	PROGRAM_COPY_MATERIAL	Unique ID associated with every component. Ideally, these values will be provided and updated via interface with LOGAIS inventory software.	VARCHAR2(10)	NOT NULL
CopyMaterial	COPY_MATERIAL	a binary object. The actual material. This could be an image, a course text, etc.	LONG	NULL
CourseAbbreviation	COURSE	Course Abbreviation as used for MMS updates	VARCHAR2(20)	NOT NULL
CourseNumber	COURSE	Course number with NO	VARCHAR2(4)	NOT

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		version		NULL
Course_ID	COURSE	Course number with version number if applicable	VARCHAR2(6)	NOT NULL
Course_ID	COURSE_COPY_MATERIAL	course number associated with this material	VARCHAR2(6)	NOT NULL
Course_ID	EXAM	Course number with version number if applicable	VARCHAR2(6)	NOT NULL
Course_ID	FINANCIAL	Course number with version number if applicable	VARCHAR2(6)	NULL
CreditHours	COURSE	Credit Hours associated with the Course for transcript generation purposes. Must contain a value if course is accredited.	VARCHAR2(3)	NULL
CreditHours	PROGRAM	Credit hours associated with the program. Must contain a value if program is accredited. Used for transcript purposes.	VARCHAR2(3)	NULL
Custodian	IS_EQUIPMENT_INVENTORY	Individual responsible for equipment	VARCHAR2(30)	NOT NULL
CustomerSSN_ID	CUSTOMER	Social Security Number of each customer being served by SSD via hotline, email, regular mail, or in person.	CHAR(9)	NOT NULL
CustomerSSN_ID	ISSUE_COMPLAINT	Social Security Number of each customer being served by SSD via hotline,	CHAR(9)	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		email, regular mail, or in person.		
CustomerSSN_ID	ORDER	Social Security Number of each customer being served by SSD via hotline, email, regular mail, or in person.	CHAR(9)	NOT NULL
DCTB	MCI_PERSONNEL	Date current tour at MCI began	DATE	NOT NULL
DSNNumber	CUSTOMER	DSN telephone number where customer can be reached	CHARACTER(7)	NULL
Date	ISSUE_COMPLAINT	Date of the complaint or issue	DATE	NOT NULL
Date	ORDER	Date of the order	DATE	NOT NULL
DateClosed	COURSE	Date course closed, if any.	DATE	NULL
DateClosed	EXAM	Date this version of the exam was closed.	DATE	NULL
DateClosed	PROGRAM	Date program closed for enrollment	DATE	NULL
DateOpened	COURSE	Date course opened for enrollment	DATE	NULL
DateOpened	EXAM	Date this version of the exam was opened.	DATE	NULL
DateOpened	PROGRAM	Date program opened for enrollment	DATE	NULL
Department	FINANCIAL	department associated with line item	VARCHAR2(40)	NOT NULL
Department	MCI_PERSONNEL	MCI department	VARCHAR2(20)	NOT NULL
Department	WAREHOUSE	Department owning the warehouse	VARCHAR2(6)	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
Description	ADVERTISMENT	a description of the advertising project	VARCHAR2(2000)	NOT NULL
Description	COURSE	Course description as provided in the MCI course catalog	LONG	NULL
Description	FINANCIAL	Description of line item	VARCHAR2(2000)	NOT NULL
Description	IS_EQUIPMENT_INVENTORY	Description of equipment item	VARCHAR2(200)	NOT NULL
Description	ISSUE_COMPLAINT	Description of the complaint or issue	VARCHAR2(400)	NOT NULL
Description	TRAINING	a description of the training program	VARCHAR2(2000)	NOT NULL
DesignedFor	COURSE	Personnel for whom course is designed, as provided in the MCI course catalog.	VARCHAR2(200)	NULL
ECC	MCI_PERSONNEL	End of current Contract	DATE	NULL
ECC	MCTF_PERS	End of Current Contract of a Marine contained in the MCTFS database.	DATE	NULL
EDD	PURCHASE	expected delivery date	DATE	NOT NULL
EventCoordinator	EVENTS	Coordinator of the Event	VARCHAR2(20)	NOT NULL
EventDate	EVENTS	Date of the Event	DATE	NOT NULL
EventDesc	EVENTS	Description of the Event	VARCHAR2(2000)	NOT NULL
EventLocation	EVENTS	Location of event	VARCHAR2(50)	NOT NULL
EventName	EVENTS	Name of the Event	VARCHAR2(100)	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
EventTime	EVENTS	Time of the Event	VARCHAR2(6)	NOT NULL
Event_ID	EVENTS	Unique Number identifying the event	NUMBER	NOT NULL
Exam_ID	EXAM	Exam number generated by the system, unique to each exam version for a specified course.	NUMBER	NOT NULL
FirstName	CUSTOMER	First Name of Customer	VARCHAR(10)	NOT NULL
FirstName	MCI_PERSONNEL	First Name	VARCHAR2(10)	NOT NULL
FirstName	MCTF_PERS	First Name of a Marine contained in the MCTFS database.	VARCHAR2(10)	NOT NULL
FirstName	STUDENT	Student's first name	VARCHAR2(10)	NOT NULL
Grade	MCTF_PERS	Grade of a Marine contained in the MCTFS database.	CHAR(2)	NOT NULL
InvoiceNumber	ORDER	Number which identifies an invoice	NUMBER	NOT NULL
IssueNumber	ISSUE_COMPLAINT	Unique number identifying the complaint or issue	NUMBER	NOT NULL
JobAidDescription	JOB_AID	Narrative description of each item	VARCHAR2(600)	NOT NULL
JobAidName	JOB_AID	Name associated with each job aid as provided by the LOGAIS system.	VARCHAR2(30)	NOT NULL
JobAid_ID	JOB_AID	Unique ID associated with every job aid. Ideally, these values will	VARCHAR2(10)	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		be provided and updated via interface with LOGAIS inventory software.		
JobAid_ID	JOB_AID_COPY_MATERIAL	job aid associated with this material	VARCHAR2(10)	NOT NULL
JobDescription	MCI_PERSONNEL	Job Description	VARCHAR2(2000)	NULL
LastName	CUSTOMER	Last name of customer	VARCHAR2(20)	NOT NULL
LastName	MCI_PERSONNEL	Last name	VARCHAR2(20)	NOT NULL
LastName	MCTF_PERS	Last Name of a Marine contained in the MCTFS database.	VARCHAR2(20)	NOT NULL
LastName	STUDENT	Student's last name	VARCHAR2(20)	NOT NULL
LineItem	FINANCIAL	Unique number associated with the budget line item	NUMBER	NOT NULL
Location	IS_EQUIPMENT_INVENTORY	Location of equipment item	VARCHAR2(50)	NOT NULL
MCTFSSN_ID	MCI_PERSONNEL	Social Security Number which uniquely identifies the instance of a Marine contained in the MCTFS database.	CHAR(9)	NULL
MCTFSSN_ID	MCTF_PERS	Social Security Number which uniquely identifies the instance of a Marine contained in the MCTFS database.	CHAR(9)	NOT NULL
MCTFSSN_ID	STUDENT	Social Security Number which uniquely identifies the instance of a Marine	CHAR(9)	NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		contained in the MCTFS database.		
MOS	MCI_PERSONNEL	Primary Military Occupational Specialty (for Military members	VARCHAR2(4)	NOT NULL
MOS	MCTF_PERS	Primary MOS of a Marine contained in the MCTFS database.	CHAR(4)	NOT NULL
MOS	STUDENT	Student's primary military occupational specialty	CHAR(4)	NULL
Manager	WAREHOUSE	warehouse manager's first and last name	VARCHAR2(40)	NOT NULL
Material_ID	COPY_MATERIAL	Number which uniquely identifies the copy material	NUMBER	NOT NULL
Material_ID	COURSE_COPY_MATERIAL	Number which uniquely identifies the copy material	NUMBER	NOT NULL
Material_ID	JOB_AID_COPY_MATERIAL	Number which uniquely identifies the copy material	NUMBER	NOT NULL
Material_ID	PROGRAM_COPY_MATERIAL	Number which uniquely identifies the copy material	NUMBER	NOT NULL
MiddleInitial	CUSTOMER	Middle Initial of Customer	CHAR(1)	NULL
MiddleInitial	MCTF_PERS	Middle Initial of a Marine contained in the MCTFS database.	VARCHAR2(1)	NULL
MiddleInitial	STUDENT	Student's middle initial	CHAR(1)	NULL
Nomenclature	IS_EQUIPMENT_INVENTORY	Nomenclature of equipment item	VARCHAR2(50)	NOT NULL
NumberOfQuestions	EXAM	Number representing the number of questions	NUMBER	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		contained on the exam.		
OccFieldSpec	COURSE_DEVELOPERS	Occupational field specialty associated with a course developer	VARCHAR2(4)	NOT NULL
OnHand	COURSE	Quantity currently on hand, as provided by interface with inventory table	NUMBER	NOT NULL
OnHand	PROGRAM	Quantity currently on hand, as provided by interface with inventory table	NUMBER	NOT NULL
OrdStatDate	PURCHASE	Date of current order status	DATE	NOT NULL
OrderDate	PURCHASE	Date order was placed	DATE	NOT NULL
Order_ID	PURCHASE	Unique invoice number which identifies the order	NUMBER	NOT NULL
PDD	MCI_PERSONNEL	Projected Departure Date	DATE	NOT NULL
PassingScore	EXAM	Number representing the passing score for the exam	NUMBER	NOT NULL
Phone	WAREHOUSE	warehouse phone number	VARCHAR2(7)	NOT NULL
PlatoonCode	MCTF_PERS	Foreign Key which identifies the Platoon Code of a Marine contained in the MCTFS database.	CHAR(4)	NOT NULL
PlatoonCode	STUDENT	Student's USMC platoon code	VARCHAR2(4)	NULL
Pointer	COPY_MATERIAL	A pointer (optional) which indicates the physical	VARCHAR2(20)	NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		location of the original of the material comprises this record. If the material is too large for storage directly in the database, this pointer will indicate its location.		
ProgramAbbreviation	PROGRAM	Program Abbreviation as used for MMS updates	VARCHAR2(20)	NOT NULL
Program_ID	FINANCIAL	Program number with version number if applicable	VARCHAR2(6)	NULL
Program_ID	PROGRAM	Program number with version number if applicable	VARCHAR2(6)	NOT NULL
Program_ID	PROGRAM_COPY_MATERIAL	program number associated with this material	VARCHAR2(6)	NOT NULL
Project_ID	ADVERTISMENT	a unique id which identifies an advertising project	NUMBER	NOT NULL
Quantity	INVENTORY		NUMBER	NOT NULL
RDD	PURCHASE	required delivery date	DATE	NULL
Rank	MCI_PERSONNEL	Military Rank or Civilian pay grade	VARCHAR2(6)	NOT NULL
Rank	MCTF_PERS	Rank of a Marine contained in the MCTFS database. This is a calculated value. The procedure used to calculate it compares the Grade and MOS to	VARCHAR2(6)	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		determine the Rank.		
ReserveCredits	COURSE	Reserve credits associated with course.	CHAR(8)	NOT NULL
SSN_ID	COURSE_DEVELOPERS	social security number	VARCHAR2(9)	NOT NULL
SSN_ID	MCI_PERSONNEL	social security number	VARCHAR2(9)	NOT NULL
SSN_ID	PROGRAM_DEVELOPERS	social security number	VARCHAR2(9)	NOT NULL
SSN_ID	SSD_PERSONNEL	social security number	VARCHAR2(9)	NOT NULL
SalesRep	PURCHASE	sales rep at the supplier who took the order	VARCHAR2(40)	NULL
SchoolCode	PROGRAM	School code associated with the program. Used for MMS purposes. Relates program to a specific resident school (e.g., AWS, etc.)	CHAR(3)	NOT NULL
Section	COURSE_DEVELOPERS	Section that Developer works in	VARCHAR2(20)	NOT NULL
Section	PROGRAM_DEVELOPERS	Section that developer works in	VARCHAR2(20)	NOT NULL
Section	SSD_PERSONNEL	Section that student services department personnel works in	VARCHAR2(20)	NOT NULL
SelToGrade	MCTF_PERS	Grade to which selected of a Marine contained in the MCTFS database.	CHAR(2)	NULL
Serial_No	IS_EQUIPMENT_INVENTORY	Serial Number of equipment item	NUMBER	NOT NULL
State	CUSTOMER	State of Customer's address	CHAR(2)	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
State	MCTF_PERS	State of the Home Address for a Marine contained in the MCTFS database. Used as the mailing Address for Reservists	VARCHAR2(2)	NULL
StudentSSN_ID	STUDENT	Social security number of student	VARCHAR2(9)	NOT NULL
StudyHours	COURSE	Study Hours associated with course. Used for transcript generation purposes	VARCHAR2(3)	NULL
StudyHours	PROGRAM	Study Hours associated with program. Used for transcript generation purposes.	VARCHAR2(3)	NULL
Supplier	PURCHASE	Organization supplying the order	VARCHAR2(100)	NOT NULL
Title	COURSE	Title of the course, as defined in the course catalog	VARCHAR2(60)	NOT NULL
Title	PROGRAM	Title of the program, as defined in the course catalog	VARCHAR2(60)	NOT NULL
TrainingProg_ID	TRAINING	a unique ID which identifies a training program	NUMBER	NOT NULL
Warehouse_ID	INVENTORY	Unique Id associated with a warehouse. Since several warehouses could be located in the same building, this is not the building number	NUMBER	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
Warehouse_ID	WAREHOUSE	Unique Id associated with a warehouse. Since several warehouses could be located in the same building, this is not the building number	NUMBER	NOT NULL
Year	FINANCIAL	Budget Year	VARCHAR2(4)	NOT NULL
ZipCode	CUSTOMER	Zip Code of customer's address	VARCHAR2(9)	NOT NULL
ZipCode	MCTF_PERS	Zip Code of the Home Address for a Marine contained in the MCTFS database. Used as the mailing Address for Reservists	VARCHAR2(9)	NOT NULL
ZipCode	STUDENT	Student's Zip Code	VARCHAR2(9)	NULL

APPENDIX G. MCI ENTERPRISE LEVEL RELATIONSHIP DEFINITIONS

Parent Entity	Verb Phrase	Child Entity	Relationship Type	Relationship Definition	Nulls	Cardinality
COMPONENTS	are tracked by	INVENTORY	Identifying	components are tracked by inventory records		One-to-Zero-One-or-More
WAREHOUSE	contain	INVENTORY	Identifying	warehouses contain inventory records		One-to-Zero-One-or-More
COURSE_DEVELOPERS	develops	COMPONENTS	Many-Many	course developers develop course components		Many-to-Many
COURSE_DEVELOPERS	develops	COURSE	Many-Many	course developers develop courses		Many-to-Many
COURSE_DEVELOPERS	develops	JOB_AID	Many-Many	course developers develop job aids		Many-to-Many
PROGRAM_DEVELOPERS	develops	COMPONENTS	Many-Many	program developers develop course components		Many-to-Many
PROGRAM_DEVELOPERS	develops	COURSE	Many-Many	a program developer develops programs		Many-to-Many
PROGRAM_DEVELOPERS	develops	PROGRAM	Many-Many	program developers develop programs		Many-to-Many
CUSTOMER	generates a	ISSUE_COMPLAINT	Identifying	a customer generates an issue or complaint in the form of a call, letter, e-mail, or walk in.		One-to-Zero-One-or-More
COURSE	has	EXAM	Identifying	a course has exams		One-to-Zero-One-or-More
COURSE	has many	STUDENT	Many-Many	a course has zero, one, or many enrolled students. A student is enrolled in zero, one, or many courses		Many-to-Many
MCI_PERSONNEL	has many	EVENTS	Many-Many	An MCI personnel may be involved in many events. An event has many personnel		Many-to-Many
PROGRAM	has many	STUDENT	Many-Many	a program has zero, one, or many students. A student is enrolled in zero, one, or many programs		Many-to-Many
COURSE	is associated with	ADVERTISMENT	Many-Many	a course is associated with an advertising project		Many-to-Many
PROGRAM	is associated	ADVERTISMENT	Many-Many	a program is associated		Many-to-Many

Parent Entity	Verb Phrase	Child Entity	Relationshlp Type	Relationshlp Definition	Nulls	Cardinality
	with			with an advertising project		
ORDER	is comprised of	COMPONENTS	Many-Many	an order is comprised of components		Many-to-Many
ORDER	is comprised of	COURSE	Many-Many	an order is comprised of courses (does not include enrollments)		Many-to-Many
ORDER	is comprised of	JOB_AID	Many-Many	an order is comprised of job aids		Many-to-Many
ORDER	is comprised of	PROGRAM	Many-Many	an order is comprised of programs (does not include enrollments)		Many-to-Many
PURCHASE	is for	COMPONENTS	Many-Many	an order is for course components		Many-to-Many
COURSE	is funded by	FINANCIAL	Non-identifying	a course is funded by zero, one, or many budget items	Nulls Allowe d	Zero-or-One-to-Zero-One-or-More
PROGRAM	is funded by	FINANCIAL	Non-identifying	a program is funded by zero, one, or many budget items	Nulls Allowe d	Zero-or-One-to-Zero-One-or-More
COMPONENTS	is part of	COURSE	Many-Many	a component is part of zero, one, or many courses. A course can one or more components		Many-to-Many
COMPONENTS	is part of	EXAM	Many-Many	a component may be part of zero, one, or many exams. An exam contains components		Many-to-Many
COMPONENTS	is part of	JOB_AID	Many-Many	a component can be part of zero, one or many job aids. A job aid can contain one or more components		Many-to-Many
COMPONENTS	is part of	PROGRAM	Many-Many	a component can be part of zero, one, or many programs. A program can contain one or more components		Many-to-Many
JOB_AID	is part of	COURSE	Many-Many	a job aid can be part of zero, one, or many courses. A course may contain zero, one, or		Many-to-Many

Parent Entity	Verb Phrase	Child Entity	Relationship Type	Relationship Definition	Nulls	Cardinality
				many job aids		
PROGRAM	is part of	COURSE	Many-Many			Many-to-Many
MCTF_PERS	may be a	MCI_PERSONNEL	Non-identifying	a MCTF Marine may be an MCI employee	Nulls Allowed	Zero-or-One-to-Exactly-1
MCTF_PERS	may be a	STUDENT	Non-identifying	A MCTF personnel may be a student	Nulls Allowed	Zero-or-One-to-Exactly-1
COMPONENTS	may be composed of	COPY_MATERIAL	Identifying	A component may be comprised of written material (known as COPY_MATERIAL)		One-to-Zero-One-or-More
CUSTOMER	places an	ORDER	Identifying	a customer places an order		One-to-Zero-One-or-More
TRAINING	relates to	MCI_PERSONNEL	Many-Many	Training is provided to MCI Personnel		Many-to-Many
SSD_PERSONNEL	resolves an	ISSUE_COMPLAINT	Many-Many	an SSD employee resolves an issue or a complaint		Many-to-Many
COPY_MATERIAL	is a	COURSE_COPY_MATERIAL	Subtype			Is a
COPY_MATERIAL	is a	JOB_AID_COPY_MATERIAL	Subtype			Is a
COPY_MATERIAL	is a	PROGRAM_COPY_MATERIAL	Subtype			Is a
MCI_PERSONNEL	is a	COURSE_DEVELOPERS	Subtype			Is a
MCI_PERSONNEL	is a	PROGRAM_DEVELOPERS	Subtype			Is a
MCI_PERSONNEL	is a	SSD_PERSONNEL	Subtype			Is a

APPENDIX H. MCI ENTERPRISE LEVEL DATA MODEL

(See pocket on inside back cover.)

APPENDIX I. MCI ENTERPRISE LEVEL MATRICES

EXHIBIT 1: DATA SUBJECTS VS. ORGANIZATIONAL UNITS

Data Subject \ Organizational Unit	Headquarters	Training & Operations Department	PME Department	Occupational Specialty Department	Student Services Department	MIS Department	Logistics Department	Unit Training Representative
Advertisement *Information*		*	*	*				
Components *Information*			*	*			*	
Copy Material *Information*			*	*				
Course *Information*				*	*	*	*	*
Course Copy Material *Information*				*				
Course Developers *Information*			*	*				
Customer *Information*					*		*	*
Events *Information*	*	*	*	*	*	*	*	
Exam *Information*		*	*	*	*	*		
Financial *Information*	*	*						
Inventory *Information*							*	
IS Equipment Inventory *Information*						*		
Issue Complaint *Information*			*		*			
Job Aid *Information*				*	*		*	*
Job Aid Copy Material *Information*				*				
MCI Personnel *Information*	*	*						
MCTFS Personnel *Information*	*	*			*	*		
Order Information					*		*	*
Program *Information*			*		*	*	*	*
Program Copy Material *Information*			*					
Program Developers *Information*			*					
Purchase *Information*					*		*	
SSD Personnel *Information*					*			
Student *Information*					*	*	*	*
Training *Information*		*						
Warehouse *Information*					*		*	

APPENDIX I. MCI ENTERPRISE LEVEL MATRICES

EXHIBIT 2: DATA SUBJECTS VS. PHYSICAL LOCATIONS

Data Subject \ Location	MCI Building 1st Floor	MCI Building 2nd Floor	Marine Barracks	MCI Warehouse	Using Unit
Advertisement *Information*	*	*			
Components *Information*	*	*		*	
Copy Material *Information*	*	*			
Course *Information*	*	*			*
Course Copy Material *Information*	*				
Course Developers *Information*	*	*			
Customer *Information*		*			
Events *Information*	*	*	*	*	
Exam *Information*	*	*			*
Financial *Information*		*			
Inventory *Information*				*	
IS Equipment Inventory *Information*		*			
Issue Complaint *Information*		*			
Job Aid *Information*	*	*			*
Job Aid Copy Material *Information*	*				
MCI Personnel *Information*	*	*	*	*	
MCTFS Personnel *Information*		*	*		
Order Information		*		*	*
Program *Information*		*			*
Program Copy Material *Information*		*			
Program Developers *Information*		*			
Purchase *Information*		*		*	
SSD Personnel *Information*		*			
Student *Information*		*			*
Training *Information*		*			
Warehouse *Information*		*		*	

APPENDIX I. MCI ENTERPRISE LEVEL MATRICES

EXHIBIT 3: DATA SUBJECTS VS. FUNCTIONS (CRUD MATRIX)

(See pocket on inside back cover.)

APPENDIX J. SSD AND MIS LEVEL CANDIDATE ENTITIES

Entity Name	Table Name
ANSWER_CODE	ANS_CODE
CATEGORY_CODE	CAT_CODE
COMPONENTS	COMP
COMPONENT_LINE_ITEM	COMP_LINE_ITM
COURSE	CRS
COURSE_COMPONENTS_X	CRS_COMP_X
COURSE_LINE_ITEM	CRS_LINE_ITM
COURSE_PREREQ	CRS_PREREQ
COURSE_PROGRAM_X	CRS_PRG_X
CUSTOMER	CUST
ED_LEVEL	ED_LEV
ERROR_CODES	ERR_CODES
ERROR_LISTING	ERR_LIST
ERR_ANS_STRING	ERR_ANS_STRING
EXAM	EXAM
EXAM_COMPONENT_X	EXAM_COMP_X
GRADE	GRADE
INVOICE	INVOICE
JOB_AID	JOB_AID
JOB_AID_COMPONENTS_X	JOB_AID_COMP_X
JOB_AID_LINE_ITEM	JOB_AID_LINE_ITM
MCTF_PERS	MCTF_PERS
MULTIPLE_CHOICE_EXAM	MULT_CHOICE
NON_MARINE_STUDENT_ADDRESS	NON_MAR_ADDR
PROGRAM	PROGRAM
PROGRAM_LINE_ITEM	PRG_LINE_ITM
PROG_COMPONENT_X	PRG_COMP_X
PROG_PREREQ	PRG_PREREQ
RANK	RANK
REMEDIATION_METHOD	REM_METH
REPORTING_UNIT_CODE	RUC
SERVICE_COMPONENT	SVC_COMP
SSD_CLERK	SSD_CLERK
STATE	STATE
STATUS_CODE	STAT_CODE
STUDENT	STUDENT
STUDENT_ANSWERS	STUD_ANS
STUDENT_COURSE_EXAM_X	STUD_CRS_EXAM_X
STUDENT_COURSE_X	STUD_CRS_X
STUDENT_PROGRAM_X	STUD_PRG_X
STUD_CRS_TRAN_X	STUD_CRS_TRAN_X
STUD_PRG_TRAN_X	STUD_PRG_TRAN_X
TRANSACTION_CODES	TRANS_CODE
TRNG_NCO_LINE_ITEM	TRNG_NCO_LINE_ITEM
TRNG_NCO_MATERIAL	TRNG_NCO_MATERIAL

APPENDIX K. SSD AND MIS LEVEL ENTITY DEFINITIONS

Entity Name	Entity Definition
ANSWER_CODE	Answer Code for a Multiple Choice Exam. Look Up Table.
CATEGORY_CODE	Domain table defining the category codes for each exam. Category codes define how an exam is to be graded.
COMPONENTS	Table of all items stocked by MCI. This includes items included in courses and programs(lessons, course material, dictionaries, etc.) as well as items included in job-aids (FAC manuals, etc.) This table will be populated via interface with LOGAIS database
COMPONENT_LINE_ITEM	An instance of a course component being ordered off line. This includes any course component requisition not associated with an enrollment.
COURSE	All active, closed, and future courses as entered by the course developer.
COURSE_COMPONENTS_X	An intersection table defining the components belonging to a specific course.
COURSE_LINE_ITEM	An instance of a course being ordered off line. This includes any course requisition not associated with an enrollment.
COURSE_PREREQ	This defines the recursive relationship between a senior course and an associated prerequisite (junior) course.
COURSE_PROGRAM_X	The intersection table defining all courses associated with a particular PME program.
CUSTOMER	Anyone who contacts MCI with a question. Contact may be by telephone, e-mail, in person, regular mail, etc.
ED_LEVEL	Domain table of all education levels. An education level is associated with a particular course for accreditation purposes.
ERROR_CODES	A domain table of all error codes associated with the automated exam scanner (Scantron).
ERROR_LISTING	A Listing of all errors by error code as generated by the automated exam scanner (Scantron).
ERR_ANS_STRING	Table containing answer strings associated with error listings
EXAM	An instance of a specific exam as generated for a specific course
EXAM_COMPONENT_X	the record that defines which course components comprise specific exams
GRADE	Domain Table identifying all particular grades associated with military personnel
INVOICE	Invoice containing offline requests for any material.
JOB_AID	Table of all job aids stocked by MCI. This table will be populated via interface with LOGAIS database
JOB_AID_COMPONENTS_X	An intersection table defining the components belonging to a specific job aid.

Entity Name	Entity Definition
JOB_AID_LINE_ITEM	An instance of a job aid being ordered off line. This includes any job aid requisition.
MCTF_PERS	Table of all MCTF personnel as downloaded from the Total Force database
MULTIPLE_CHOICE_EXAM	The intersection table defining the questions associated with a specific exam
NON_MARINE_STUDENT_ADDRESS	Any Student Not contained in the MCTFS database
PROGRAM	All active, closed, and future programs as entered by the program developer.
PROGRAM_LINE_ITEM	An instance of a program being ordered off line. This includes any program requisition not associated with an enrollment.
PROG_COMPONENT_X	Defines those components associated directly with programs, which are not associated with the sub-courses of a program
PROG_PREREQ	This defines the recursive relationship between a senior program and an associated prerequisite (junior) program.
RANK	Domain Table identifying all particular ranks associated with military personnel
REMEDIATION_METHOD	Domain table to store remediation methods for exams
REPORTING_UNIT_CODE	Table of all Reporting Unit Codes as downloaded from the Total Force database
SERVICE_COMPONENT	Domain table defining the component codes for each student. Component codes define a students service affiliation.
SSD_CLERK	Information on hotline operator, or MCI student services representative handling any issue generated by media besides hotline (e-mail, regular mail, etc.)
STATE	Domain table of all states, territories, and the District of Columbia
STATUS_CODE	Domain table listing all possible course status codes and their definitions
STUDENT	All Students contained in the MCI database
STUDENT_ANSWERS	contains an array of student's answers to a multiple choice exam
STUDENT_COURSE_EXAM_X	Intersection table of all exams issued to an instance of a student enrolled in a course
STUDENT_COURSE_X	An instance of a student enrolled in a course.
STUDENT_PROGRAM_X	An instance of a student enrolled in a program
STUD_CRS_TRAN_X	Intersection table recording all transactions performed on an instance of a student enrolled in a course
STUD_PRG_TRAN_X	Intersection table recording all transactions performed on an instance of a student enrolled in a course
TRANSACTION_CODES	Domain table of all transactions which may be conducted on an instance of a student enrolled in either a course or a program
TRNG_NCO_LINE_ITEM	An instance of training NCO material being ordered off line. This includes any training NCO material

Entity Name	Entity Definition
	requisition.
TRNG_NCO_MATERIAL	Table containing instances of material which is ordered by unit training NCOs.

Entity Name	Entity Type	Primary Key	Foreign Keys	Parent In Relationships
ANSWER_CODE	Independent	AnsCode_ID		ANSWER_CODE has a MULTIPLE_CHOICE_EXAM Cardinality: One-to-Zero-One-or-More No Null
CATEGORY_CODE	Independent	CatCode_ID		CATEGORY_CODE Defines Grading Method EXAM Cardinality: One-to-Zero-One-or-More No Null
COMPONENTS	Independent	Comp_ID		COMPONENTS may be included in PROG_COMPONENT_X Cardinality: One-to-Zero-One-or-More COMPONENTS comprises a EXAM_COMPONENT_X Cardinality: One-to-Zero-One-or-More COMPONENTS comprises a JOB_AID_COMPONENTS_X Cardinality: One-to-Zero-One-or-More COMPONENTS Is ordered as COMPONENT_LINE_ITEM Cardinality: One-to-Zero-One-or-More COMPONENTS comprises a COURSE_COMPONENTS_X Cardinality: One-to-Zero-One-or-More
COMPONENT_LINE_ITEM	Dependent	InvNo_ID + Comp_ID	Component_ID References: COMPONENTS + InvoiceNumber_ID References: INVOICE	
COURSE	Independent	Crs_ID	RemediationMeth References: REMEDIATION_METHOD + StatusCode References:	COURSE has a prereq of COURSE_PREREQ Cardinality: One-to-Zero-One-or-More COURSE is a prereq for COURSE_PREREQ Cardinality: One-to-Zero-One-or-More COURSE Is ordered as

Entity Name	Entity Type	Primary Key	Foreign Keys	Parent In Relationships
			STATUS_CODE + EdLevel References: ED_LEVEL	COURSE_LINE_ITEM Cardinality: One-to-Zero-One-or-More COURSE is part of COURSE_PROGRAM_X Cardinality: One-to-Zero-One-or-More COURSE Is comprised of COURSE_COMPONENTS_X Cardinality: One-to-Zero-One-or-More COURSE Evaluated-By EXAM Cardinality: One-to-Zero-One-or-More COURSE Contains STUDENT_COURSE_X Cardinality: One-to-Zero-One-or-More
COURSE_COMPONENTS_X	Dependent	Crs_ID + Comp_ID	Course_ID References: COURSE + Component_ID References: COMPONENTS	
COURSE_LINE_ITEM	Dependent	InvNo_ID + Crs_ID	InvoiceNumber_ID References: INVOICE + Course_ID References: COURSE	
COURSE_PREREQ	Dependent	JRCRS_ID + SRCRS_ID	JrCrs_ID References: COURSE + SrCrs_ID References: COURSE	
COURSE_PROGRAM_X	Dependent	Crs_ID + Prg_ID	Program_ID References: PROGRAM + Course_ID	

Entity Name	Entity Type	Primary Key	Foreign Keys	Parent In Relationships
			References: COURSE	
CUSTOMER	Independent	CustSSn_ID	State_ID References: STATE	CUSTOMER is associated with INVOICE Cardinality: One-to-One-or-More (P) No Null
ED_LEVEL	Independent	EdLev		ED_LEVEL is associated with COURSE Cardinality: Zero-or-One-to-Zero-One-or-More ED_LEVEL Is associated with PROGRAM Cardinality: Zero-or-One-to-Zero-One-or-More
ERROR_CODES	Independent	ErrCode_ID		ERROR_CODES Is contained on ERROR_LISTING Cardinality: One-to-Zero-One-or-More No Null
ERROR_LISTING	Independent	ErrCntr_ID + ErrDate	ErrorCode_ID References: ERROR_CODES + StudentSSN_ID, Exam_ID, Course_ID References: STUDENT_COURSE_EXAM_X	ERROR_LISTING have associated ERR_ANS_STRING Cardinality: One-to-Zero-One-or-More
ERR_ANS_STRING	Dependent	ErrCntr_ID + ErrDate	ErrorCounter_ID, ErrorDate References: ERROR_LISTING	
EXAM	Dependent	Exam_ID + Crs_ID	CategoryCode_ID References: CATEGORY_CODE + Course_ID References: COURSE	EXAM is comprised of EXAM_COMPONENT_X Cardinality: One-to-Zero-One-or-More EXAM Belongs to STUDENT_COURSE_EXAM_X Cardinality: One-to-Zero-One-or-

Entity Name	Entity Type	Primary Key	Foreign Keys	Parent In Relationships
				More MULTIPLE_CHOICE_EXAM is subcategory of EXAM
EXAM_COMPONENT_X	Dependent	Comp_ID + Exam_ID + Crs_ID	Exam_ID, Course_ID References: EXAM + Component_ID References: COMPONENTS	
GRADE	Independent	Grade		GRADE defines grade of STUDENT Cardinality: Zero-or-One-to-Zero-One-or-More
INVOICE	Independent	InvNo_ID	CustSSn_ID References: CUSTOMER + ClerkSSN_ID References: SSD_CLERK	INVOICE contains TRNG_NCO_LINE_ITEM Cardinality: One-to-Zero-One-or-More INVOICE contains JOB_AID_LINE_ITEM Cardinality: One-to-Zero-One-or-More INVOICE contains PROGRAM_LINE_ITEM Cardinality: One-to-Zero-One-or-More INVOICE contains COURSE_LINE_ITEM Cardinality: One-to-Zero-One-or-More INVOICE contains COMPONENT_LINE_ITEM Cardinality: One-to-Zero-One-or-More
JOB_AID	Independent	JobAid_ID		JOB_AID is ordered as JOB_AID_LINE_ITEM Cardinality: One-to-Zero-One-or-More JOB_AID is comprised of JOB_AID_COMPONENTS_X Cardinality: One-to-Zero-One-or-More

Entity Name	Entity Type	Primary Key	Foreign Keys	Parent In Relationships
JOB_AID_COMPONENTS_X	Dependent	Comp_ID + JobAid_ID	JobAid_ID References: JOB_AID + Component_ID References: COMPONENTS	
JOB_AID_LINE_ITEM	Dependent	InvNo_ID + JobAid_ID	JobAid_ID References: JOB_AID + InvoiceNumber_ID References: INVOICE	
MCTF_PERS	Independent	MCTFSSN	MCTF_RUCMCC References: REPORTING_UNIT_CODE	
MULTIPLE_CHOICE_EXAM	Dependent	QuestNo + Exam_ID + Crs_ID	Exam_ID, Course_ID References: EXAM + Answer_Code_ID References: ANSWER_CODE	
NON_MARINE_STUDENT_ADDRESS	Dependent	StudSSN_ID	StudentSSN_ID References: STUDENT + State_ID References: STATE	
PROGRAM	Independent	Prg_ID	RemediationMeth References: REMEDIATION_METHOD + StatusCode References:	PROGRAM may consist of PROG_COMPONENT_X Cardinality: One-to-Zero-One-or-More PROGRAM Is a prerequisite for PROG_PREREQ Cardinality: One-to-Zero-One-or-More

Entity Name	Entity Type	Primary Key	Foreign Keys	Parent In Relationships
			STATUS_CODE + EdLevel_FK References: ED_LEVEL	PROGRAM Has a prerequisite of PROG_PREREQ Cardinality: One-to-Zero-One-or-More PROGRAM Is ordered as PROGRAM_LINE_ITEM Cardinality: One-to-Zero-One-or-More PROGRAM is made up of COURSE_PROGRAM_X Cardinality: One-to-Zero-One-or-More PROGRAM Contains STUDENT_PROGRAM_X Cardinality: One-to-Zero-One-or-More
PROGRAM_LINE_ITEM	Dependent	InvNo_ID + Prg_ID	Program_ID References: PROGRAM + InvoiceNumber_ID References: INVOICE	
PROG_COMPONENT_X	Dependent	Prg_ID + Comp_ID	Component_ID References: COMPONENTS + Program_ID References: PROGRAM	
PROG_PREREQ	Dependent	SrProg_ID + JrProg_ID	SrProg__ID References: PROGRAM + JrProg_ID References: PROGRAM	
RANK	Independent	Rank		RANK defines rank of STUDENT Cardinality: Zero-or-One-to-Zero-One-or-More
REMEDIATION_METHOD	Independent	RemMeth_ID		REMEDIATION_METHOD is associated with COURSE

Entity Name	Entity Type	Primary Key	Foreign Keys	Parent In Relationships
				Cardinality: One-to-Zero-One-or-More No Null REMEDIATION_METHOD is associated with PROGRAM Cardinality: One-to-Zero-One-or-More No Null
REPORTING_UNIT_CODE	Independent	MCTF_RUCMCC		REPORTING_UNIT_CODE has associated MCTF_PERS Cardinality: One-to-Zero-One-or-More No Null
SERVICE_COMPONENT	Independent	ServComp		SERVICE_COMPONENT Defines Service Affiliation of STUDENT Cardinality: One-to-Zero-One-or-More No Null
SSD_CLERK	Independent	ClerkSSN_ID		SSD_CLERK opens an INVOICE Cardinality: Zero-or-One-to-Zero-One-or-More
STATE	Independent	State		STATE defines State of CUSTOMER Cardinality: Zero-or-One-to-Zero-One-or-More STATE defines state of NON_MARINE_STUDENT_ADDRESS Cardinality: Zero-or-One-to-Zero-One-or-More
STATUS_CODE	Independent	StatCode_ID		STATUS_CODE Defines status of COURSE Cardinality: One-to-Zero-One-or-More No Null STATUS_CODE Defines Status Of PROGRAM Cardinality: One-to-Zero-One-or-More No Null
STUDENT	Independent	StudSSN_ID	Rank_ID References: RANK + Grade References: GRADE + ServiceComponent	STUDENT may have an NON_MARINE_STUDENT_ADDRESS Cardinality: One-to-Zero-or-One (Z) STUDENT Is enrolled in STUDENT_COURSE_X Cardinality: One-to-Zero-One-or-

Entity Name	Entity Type	Primary Key	Foreign Keys	Parent In Relationships
			_ID References: SERVICE_COMPONENT	More STUDENT Is issued a STUDENT_COURSE_EXAM_X Cardinality: One-to-Zero-One-or-More STUDENT Is enrolled in STUDENT_PROGRAM_X Cardinality: One-to-Zero-One-or-More
STUDENT_ANSWERS	Dependent	StudSSN_ID + Exam_ID + Crs_ID	StudentSSN_ID, Exam_ID, Course_ID References: STUDENT_COURSE_EXAM_X	
STUDENT_COURSE_EXAM_X	Dependent	StudSSN_ID + Exam_ID + Crs_ID	Exam_ID, Course_ID References: EXAM + StudentSSN_ID References: STUDENT	STUDENT_COURSE_EXAM_X have associated STUDENT_ANSWERS Cardinality: One-to-Zero-One-or-More STUDENT_COURSE_EXAM_X contains ERROR_LISTING Cardinality: Zero-or-One-to-Zero-One-or-More
STUDENT_COURSE_X	Dependent	Crs_ID + StudSSN_ID	StudentSSN_ID References: STUDENT + Course_ID References: COURSE	STUDENT_COURSE_X has an associated STUD_CRS_TRAN_X Cardinality: One-to-Zero-One-or-More
STUDENT_PROGRAM_X	Dependent	Prg_ID + StudSSN_ID	Program_ID References: PROGRAM + StudentSSN_ID References: STUDENT	STUDENT_PROGRAM_X has an associated STUD_PRG_TRAN_X Cardinality: One-to-Zero-One-or-More
STUD_CRS_TRAN_X	Dependent	Crs_ID + StudSSN_ID +	Transaction_ID References:	

Entity Name	Entity Type	Primary Key	Foreign Keys	Parent In Relationships
		Tran_ID + SC_TransDate_ ID	TRANSACTION_ CODES + Course_ID, StudentSSN_ID References: STUDENT_COU RSE_X	
STUD_PRG_TRAN_X	Dependent	Tran_ID + Prg_ID + StudSSN_ID + SP_TransDate_ ID	Transaction_ID References: TRANSACTION_ CODES + Program_ID, StudentSSN_ID References: STUDENT_PRO GRAM_X	
TRANSACTION_CODES	Independent	Tran_ID		TRANSACTION_CODES associated with STUD_CRS_TRAN_X Cardinality: One-to-Zero-One-or-More TRANSACTION_CODES associated with STUD_PRG_TRAN_X Cardinality: One-to-Zero-One-or-More
TRNG_NCO_LINE_ITEM	Dependent	InvNo_ID + Item_ID	Item_ID TRNG_NCO_MA TERIAL + InvoiceNumber_I D References: INVOICE	
TRNG_NCO_MATERIAL	Independent	Item_ID		TRNG_NCO_MATERIAL is ordered as TRNG_NCO_LINE_ITEM Cardinality: One-to-Zero-One-or-More

APPENDIX M. SSD AND MIS LEVEL CANDIDATE ATTRIBUTES

Attribute Name	Column Name
Answer_Code_ID	AnsCode_ID
Answer_Code_ID	AnsCode_ID
AreasOfStudy	AreasOfStudy
C_Accredited	C_Accr
C_GradePrereq	C_GradePre
C_PME	C_PME
CategoryCode_ID	CatCode_ID
CategoryCode_ID	CatCode_ID
ClerkFirstName	ClerkFirstName
ClerkLastName	ClerkLastName
ClerkMiddleInitial	ClerkMidInit
ClerkSSN_ID	ClerkSSN_ID
ClerkSSN_ID	ClerkSSN_ID
CodeDescription	AnsCodeDesc
CompOnHandQty	CompOnHandQty
ComponentDescription	CompDesc
ComponentName	CompName
ComponentQuantity	CompQty
Component_ID	Comp_ID
Component_ID	Comp_ID
Component_ID	Comp_ID
Component_ID	Comp_ID
Component_ID	Comp_ID
Component_ID	Comp_ID
CourseAbbreviation	CrsAbbr
CourseAdministration	CourseAdministration
CourseDescription	CrsDesc
CourseDesignedFor	CrsDesignedFor
CourseDeveloper	CourseDeveloper
CourseNumber	CrsNo
CourseTitle	CrsTitle
Course_ID	Crs_ID
Course_ID	Crs_ID
Course_ID	Crs_ID
Course_ID	Crs_ID
Course_ID	Crs_ID
Course_ID	Crs_ID
Course_ID	Crs_ID
Course_ID	Crs_ID
Course_ID	Crs_ID
Course_ID	Crs_ID
Course_ID	Crs_ID
Course_ID	Crs_ID
Course_ID	ScanCrs_ID
CrsCCD	CrsCCD
CrsCertificateIssueDate	CrCertDate
CrsCreditHours	CrsCreditHrs
CrsDateClosed	CrsDateClsd
CrsDateOpened	CrsDateOpen

Attribute Name	Column Name
CrsEnrollDate	CrsEnrollDate
CrsFinalGrade	CrsFinalGrade
CrsOnHandQty	CrsOnHandQty
CrsQuanity	CrsQty
CrsReserveCredits	CrsResCred
CrsStudyHours	CrsStudyHours
CustAddr1	CustAddr1
CustAddr2	CustAddr2
CustCity	CustCity
CustCommNo	CustCommNo
CustDSN	CustDSN
CustFirstName	CustFirstName
CustLastName	CustLastName
CustMidInit	CustMidInit
CustSSn_ID	CustSSn_ID
CustSSn_ID	CustSSn_ID
CustZip	CustZip
Description	CatCodeDesc
Disenrolled	Disenrolled
Disenrolled	Disenrolled
EDLevel_ID	EdLev
EdLevDescription	EdLevDesc
EdLevel	EdLev
EdLevel_FK	EdLev
ErrorCodeDescription	ErrCodeDesc
ErrorCode_ID	ErrCode_ID
ErrorCode_ID	ErrCode_ID
ErrorCounter_ID	ErrCntr_ID
ErrorCounter_ID	ErrCntr_ID
ErrorDate	ErrDate
ErrorDate	ErrDate
ExCompQuantity	ExmCompQty
ExamAnswerString	ExmAnsStrng
ExamDateClosed	ExmDateClose
ExamDateOpened	ExmDateOpen
ExamDateReceived	ExmDateRecv
ExamNumberOfQuestions	ExNoQuest
ExamPassingScore	ExmPassScore
ExamReference_1	ExamRef_1
ExamReference_2	ExamRef_2
ExamScore	ExmScore
Exam_ID	Exam_ID
Exam_ID	Exam_ID
Exam_ID	Exam_ID
Exam_ID	Exam_ID
Exam_ID	Exam_ID
Exam_ID	ScanExam_ID
ExmAnsStrng	ExmAnsStrng
Grade	Grade
Grade	Grade

Attribute Name	Column Name
GradeDescription	GradeDesc
InvDate	InvDate
InvoiceNumber_ID	InvNo_ID
InvoiceNumber_ID	InvNo_ID
InvoiceNumber_ID	InvNo_ID
InvoiceNumber_ID	InvNo_ID
InvoiceNumber_ID	InvNo_ID
InvoiceNumber_ID	InvNo_ID
ItemDescription	ItemDescription
ItemName	ItemName
ItemQty	ItemQty
Item_ID	Item_ID
Item_ID	Item_ID
JACompQuantity	JACompQty
JAQuantity	JAQty
JobAidDescription	JADesc
JobAidName	JAName
JobAidOnHandQty	JAOnHandQty
JobAid_ID	JobAid_ID
JobAid_ID	JobAid_ID
JobAid_ID	JobAid_ID
JrCrs_ID	JRCRS_ID
JrProg_ID	JrProg_ID
MCTFAddr1	MCTFAddr1
MCTFAddr2	MCTFAddr2
MCTFBilletMOS	MCTFBilletMOS
MCTFCity	MCTFCity
MCTFComponent	MCTFComp
MCTFFirstInitial	MCTFFirstInitial
MCTFGrade	MCTFGrade
MCTFLastName	MCTFLastName
MCTFMOS	MCTFMOS
MCTFMiddleInitial	MCTFMidInit
MCTFPlatoonCode	MCTFPlatCode
MCTFRank	MCTFRank
MCTFSSN_ID	MCTFSSN
MCTFSelToGrade	MCTFSelToGrade
MCTFZipCode	MCTFZip
MCTF_ECC	MCTF_ECC
MCTF_RUCMCC	MCTF_RUCMCC
NMSAddr1	NMSAddr1
NMSAddr2	NMSAddr2
NMSCity	NMSCity
NMSZipCode	NMSZip
OpscanNumber	OpscanNumber
P_Accredited	P_Accr
P_GradeElOther	P_GradeElOther
P_GradeElUSMC	P_GradeElUSMC
Prerequisites	Prerequisites
PrgCCD	PrgCCD

Attribute Name	Column Name
PrgCompQty	PrgCompQty
PrgCreditHours	PrgCreditHrs
PrgDateClosed	PrgDateClosed
PrgDateOpened	PrgDateOpened
PrgDiplomaIssueDate	PrgDipIssueDate
PrgEnrollDate	PrgEnrollDate
PrgOnHandQty	PrgOnHandQty
PrgQuantity	PrgQty
PrgStudyHours	PrgStudyHrs
PrgTitle	PrgTitle
ProgramAbbreviation	ProgAbbr
Program_ID	Prg_ID
Program_ID	Prg_ID
Program_ID	Prg_ID
Program_ID	Prg_ID
Program_ID	Prg_ID
Program_ID	Prg_ID
Purpose	Purpose
QtyOnHand	QtyOnHand
Quantity	CC_Qty
QuestionNumber	QuestNo
RUCAddr1	RUCAddr1
RUCAddr2	RUCAddr2
RUCCityState	RUCCityState
RUCMCC	MCTF_RUCMCC
RUCZip	RUCZip
RankDescription	RankDesc
Rank_ID	Rank
Rank_ID	Rank
RemMethDescription	RemMethDesc
RemediationMeth	RemMeth_ID
RemediationMeth	RemMeth_ID
RemediationMeth	RemMeth_ID
SC_Comp	SC_Comp
SC_PME	SC_PME
SC_TransDate_ID	SC_TransDate_ID
SP_Comp	SP_Comp
SP_TransDate_ID	SP_TransDate_ID
SchoolCode	SchoolCode
ServCompDescription	ServCompDesc
ServiceComponent_ID	ServComp
ServiceComponent_ID	ServComp
SrCrs_ID	SRCRS_ID
SrProg__ID	SrProg_ID
State	MCTFState
StateDescription	StateDesc
State_ID	State
State_ID	State
State_ID	State
StatusCode	StatCode_ID

Attribute Name	Column Name
StatusCode	StatCode_ID
StatusCodeDescription	StatCodeDesc
StatusCode_ID	StatCode_ID
StudFirstName	StudFirstName
StudLastName	StudLastName
StudMI	StudMI
StudentSSN_ID	ScanSSN_ID
StudentSSN_ID	StudSSN_ID
StudentSSN_ID	StudSSN_ID
StudentSSN_ID	StudSSN_ID
StudentSSN_ID	StudSSN_ID
StudentSSN_ID	StudSSN_ID
StudentSSN_ID	StudSSN_ID
StudentSSN_ID	StudSSN_ID
StudentSSN_ID	StudSSN_ID
TransactionCodeDescription	TransDesc
Transaction_ID	Tran_ID
Transaction_ID	Tran_ID
Transaction_ID	Tran_ID
UnitName	UnitName

APPENDIX N. SSD AND MIS LEVEL ATTRIBUTE DEFINITIONS

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
Answer_Code_ID	ANSWER_CODE	Identifies the Answer Code for a Multiple Choice Exam	VARCHAR2(1)	NOT NULL
Answer_Code_ID	MULTIPLE_CHOICE_EXAM	Identifies the Answer Code for a Multiple Choice Exam	VARCHAR2(1)	NOT NULL
AreasOfStudy	PROGRAM	Areas studied by the program	VARCHAR2(1000)	NULL
C_Accredited	COURSE	Flag which indicates if this course is accredited	CHAR(1)	NOT NULL
C_GradePrereq	COURSE	Prerequisite grade associated with this course	CHAR(2)	NULL
C_PME	COURSE	Flag which indicates if this course is part of a PME program	CHAR(1)	NOT NULL
CategoryCode_ID	CATEGORY_CODE	Alpha-numeric code for how to grade exam. Defined by corresponding Description	VARCHAR2(2)	NOT NULL
CategoryCode_ID	EXAM	Alpha-numeric code for how to grade exam. Defined by corresponding Description	VARCHAR2(2)	NOT NULL
ClerkFirstName	SSD_CLERK	First name of the MCI student services representative handling the issue	CHAR(1)	NOT NULL
ClerkLastName	SSD_CLERK	Last name of MCI student services	VARCHAR2(20)	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		representative handling the issue		
ClerkMiddleInitial	SSD_CLERK	Middle initial of the MCI student services representative handling the issue	CHAR(1)	NOT NULL
ClerkSSN_ID	INVOICE	Social Security Number of the MCI student services representative handling the order	CHAR(9)	NULL
ClerkSSN_ID	SSD_CLERK	Social Security Number of the MCI student services representative handling the order	CHAR(9)	NOT NULL
CodeDescription	ANSWER_CODE	Describes the Answer Code for a Multiple Choice Exam	VARCHAR2(30)	NOT NULL
CompOnHandQty	COMPONENTS	Indicates on hand quantity of the component	NUMBER	NOT NULL
ComponentDescription	COMPONENTS	Narrative description of each item	VARCHAR2(65)	NOT NULL
ComponentName	COMPONENTS	Name associated with each component as provided by the LOGAIS system.	VARCHAR2(65)	NOT NULL
ComponentQuantity	COMPONENT_LINE_ITEM	Quantity of this component requested.	NUMBER	NOT NULL
Component_ID	COMPONENTS	Unique ID associated with every course component. Ideally, these values will be provided and updated	VARCHAR2(10)	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		via interface with LOGAIS inventory software.		
Component_ID	COMPONENT_LINE_ITEM	Unique ID associated with every course component. Ideally, these values will be provided and updated via interface with LOGAIS inventory software.	VARCHAR2(10)	NOT NULL
Component_ID	COURSE_COMPONENTS_X	Unique ID associated with every course component. Ideally, these values will be provided and updated via interface with LOGAIS inventory software.	VARCHAR2(10)	NOT NULL
Component_ID	EXAM_COMPONENT_X	Unique ID associated with every course component. Ideally, these values will be provided and updated via interface with LOGAIS inventory software.	VARCHAR2(10)	NOT NULL
Component_ID	JOB_AID_COMPONENTS_X	Unique ID associated with every course component. Ideally, these values will be provided and updated via interface with LOGAIS inventory	VARCHAR2(10)	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		software.		
Component_ID	PROG_COMPONENT_X	Unique ID associated with every course component. Ideally, these values will be provided and updated via interface with LOGAIS inventory software.	VARCHAR2(10)	NOT NULL
CourseAbbreviation	COURSE	Course Abbreviation as used for MMS updates	VARCHAR2(20)	NOT NULL
CourseAdministration	PROGRAM	How the program will be administered	VARCHAR2(2000)	NULL
CourseDescription	COURSE	Course description as provided in the MCI course catalog	LONG	NULL
CourseDesignedFor	COURSE	Personnel for whom course is designed, as provided in the MCI course catalog.	VARCHAR2(200)	NULL
CourseDeveloper	PROGRAM	Phone number of course developer	VARCHAR2(50)	NULL
CourseNumber	COURSE	Course number with NO version	VARCHAR2(4)	NOT NULL
CourseTitle	COURSE	Title of the course, as defined in the course catalog	VARCHAR2(60)	NOT NULL
Course_ID	COURSE	Course number with version number if applicable	VARCHAR2(6)	NOT NULL
Course_ID	COURSE_COMPONENTS_X	Course number with version number if applicable	VARCHAR2(6)	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
Course_ID	COURSE_LINE_ITEM	Course number with version number if applicable	VARCHAR2(6)	NOT NULL
Course_ID	COURSE_PROGRAM_X	Course number with version number if applicable	VARCHAR2(6)	NOT NULL
Course_ID	ERROR_LISTING	Course number with version number if applicable	VARCHAR2(6)	NULL
Course_ID	EXAM	Course number with version number if applicable	VARCHAR2(6)	NOT NULL
Course_ID	EXAM_COMPONENT_X	Course number with version number if applicable	VARCHAR2(6)	NOT NULL
Course_ID	MULTIPLE_CHOICE_EXAM	Course number with version number if applicable	VARCHAR2(6)	NOT NULL
Course_ID	STUDENT_ANSWERS	Course number with version number if applicable	VARCHAR2(6)	NOT NULL
Course_ID	STUDENT_COURSE_EXAM_X	Course number with version number if applicable	VARCHAR2(6)	NOT NULL
Course_ID	STUDENT_COURSE_X	Course number with version number if applicable	VARCHAR2(6)	NOT NULL
Course_ID	STUD_CRS_TRAN_X	Course number with version number if applicable	VARCHAR2(6)	NOT NULL
CrsCCD	STUDENT_COURSE_X	Course Completion Deadline	DATE	NOT NULL
CrsCertificateIssueDate	STUDENT_COURSE_X	Date certificate for	DATE	NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		course completion is issued to the student.		
CrsCreditHours	COURSE	Credit Hours associated with the Course for transcript generation purposes. Must contain a value if course is accredited.	VARCHAR2(3)	NULL
CrsDateClosed	COURSE	Date course closed, if any.	DATE	NULL
CrsDateOpened	COURSE	Date course opened for enrollment	DATE	NULL
CrsEnrollDate	STUDENT_COURSE_X	Date student was enrolled in the course	DATE	NOT NULL
CrsFinalGrade	STUDENT_COURSE_X	Student's final course grade.	VARCHAR2(3)	NULL
CrsOnHandQty	COURSE	Quantity currently on hand, as provided by interface with inventory table	NUMBER	NOT NULL
CrsQuanity	COURSE_LINE_ITEM	Quantity of this course requested.	NUMBER	NOT NULL
CrsReserveCredits	COURSE	Reserve credits associated with course.	VARCHAR2(3)	NULL
CrsStudyHours	COURSE	Study Hours associated with course. Used for transcript generation purposes	NUMBER	NULL
CustAddr1	CUSTOMER	First line of address of customer	VARCHAR2(46)	NOT NULL
CustAddr2	CUSTOMER	Second line of address of customer	VARCHAR2(46)	NULL
CustCity	CUSTOMER	City of customer	VARCHAR2(34)	NOT

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
				NULL
CustCommNo	CUSTOMER	Commercial phone number of customer	VARCHAR2(14)	NULL
CustDSN	CUSTOMER	DSN phone number of customer	VARCHAR2(11)	NULL
CustFirstName	CUSTOMER	First name of customer	VARCHAR2(10)	NOT NULL
CustLastName	CUSTOMER	Last name of customer	VARCHAR2(20)	NOT NULL
CustMidInit	CUSTOMER	Middle Initial of customer	CHAR(1)	NULL
CustSSn_ID	CUSTOMER	Social Security Number of each customer being served by SSD via hotline, email, regular mail, or in person.	VARCHAR2(9)	NOT NULL
CustSSn_ID	INVOICE	Social Security Number of each customer being served by SSD via hotline, email, regular mail, or in person.	CHAR(9)	NOT NULL
CustZip	CUSTOMER	Zip of customer	VARCHAR2(9)	NOT NULL
Description	CATEGORY_CODE	Defines how exam will be graded	VARCHAR2(20)	NOT NULL
Disenrolled	STUDENT_COURSE_X	Flag which indicates if a student has been disenrolled from a course	CHAR(1)	NOT NULL
Disenrolled	STUDENT_PROGRAM_X	Flag which indicates if a student has been disenrolled from a program	CHAR(1)	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
EDLevel_ID	ED_LEVEL	The education level associated with a Course or Program	VARCHAR2(4)	NOT NULL
EdLevDescription	ED_LEVEL	Description of each education level	VARCHAR2(20)	NOT NULL
EdLevel	COURSE	The education level associated with a Course or Program	VARCHAR2(4)	NULL
EdLevel_FK	PROGRAM	The education level associated with a Course or Program	VARCHAR2(4)	NULL
ErrorCodeDescription	ERROR_CODES	a narrative description of the error code	VARCHAR2(55)	NOT NULL
ErrorCode_ID	ERROR_CODES	Identifier of the error codes available in the domain table containing error codes generated during automatic reading of exam answer sheets (error codes available to the SCANTRON reader).	VARCHAR2(1)	NOT NULL
ErrorCode_ID	ERROR_LISTING	Identifier of the error codes available in the domain table containing error codes generated during automatic reading of exam answer sheets (error codes available to the SCANTRON reader).	VARCHAR2(1)	NOT NULL
ErrorCounter_ID	ERROR_LISTING	Counter associated with the error code. This	NUMBER	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		counter is associated with a specific date, is reset automatically to zero each day at time 0000, and automatically increments for each error generated.		
ErrorCounter_ID	ERR_ANS_STRING	Counter associated with the error code. This counter is associated with a specific date, is reset automatically to zero each day at time 0000, and automatically increments for each error generated.	NUMBER	NOT NULL
ErrorDate	ERROR_LISTING	The date of the error, as read by the system clock	DATE	NOT NULL
ErrorDate	ERR_ANS_STRING	The date of the error, as read by the system clock	DATE	NOT NULL
ExCompQuantity	EXAM_COMPONENT_X	Quantity of this component required for this instance	NUMBER	NOT NULL
ExamAnswerString	STUDENT_ANSWERS	Answer String of student's answers (0-160) on a multiple choice exam. Spaces are present for NULL answer fields.	VARCHAR2(160)	NOT NULL
ExamDateClosed	EXAM	Date this version of the exam was closed.	DATE	NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
ExamDateOpened	EXAM	Date this version of the exam was opened.	DATE	NULL
ExamDateReceived	STUDENT_COURSE_EXAM_X	Date exam received at MCI	DATE	NULL
ExamNumberOfQuestions	EXAM	Number representing the number of questions contained on the exam.	NUMBER	NOT NULL
ExamPassingScore	EXAM	Number representing the passing score for the exam	NUMBER	NOT NULL
ExamReference_1	MULTIPLE_CHOICE_EXAM	Primary reference for exam question from the course material	VARCHAR2(10)	NOT NULL
ExamReference_2	MULTIPLE_CHOICE_EXAM	Secondary reference for exam question from the course material	VARCHAR2(10)	NULL
ExamScore	STUDENT_COURSE_EXAM_X	Student's exam score	NUMBER	NULL
Exam_ID	ERROR_LISTING	Exam number generated by the system, unique to each exam version for a specified course.	VARCHAR2(2)	NULL
Exam_ID	EXAM	Exam number generated by the system, unique to each exam version for a specified course.	VARCHAR2(2)	NOT NULL
Exam_ID	EXAM_COMPONENT_X	Exam number generated by the system, unique to each exam version for a specified course.	VARCHAR2(2)	NOT NULL
Exam_ID	MULTIPLE_CHOICE_EXAM	Exam number generated by the system, unique to each exam version for a specified course.	VARCHAR2(2)	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
Exam_ID	STUDENT_ANSWERS	Exam number generated by the system, unique to each exam version for a specified course.	VARCHAR2(2)	NOT NULL
Exam_ID	STUDENT_COURSE_EXAM_X	Exam number generated by the system, unique to each exam version for a specified course.	VARCHAR2(2)	NOT NULL
ExmAnsStrng	ERR_ANS_STRING	Answer string associated with the error listing	VARCHAR2(160)	NOT NULL
Grade	GRADE	Abbreviation normally associated with a specific grade for a specific military service or services.	CHAR(2)	NOT NULL
Grade	STUDENT	Abbreviation normally associated with a specific grade for a specific military service or services.	CHAR(2)	NULL
GradeDescription	GRADE	Description of the Grade abbreviation contained in the primary key. Describes the grade and its service affiliation	VARCHAR2(20)	NOT NULL
InvDate	INVOICE	Date of Invoice	DATE	NOT NULL
InvoiceNumber_ID	COMPONENT_LINE_ITEM	System assigned invoice number which uniquely identifies each off line order invoice.	NUMBER	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
InvoiceNumber_ID	COURSE_LINE_ITEM	System assigned invoice number which uniquely identifies each off line order invoice.	NUMBER	NOT NULL
InvoiceNumber_ID	INVOICE	System assigned invoice number which uniquely identifies each off line order invoice.	NUMBER	NOT NULL
InvoiceNumber_ID	JOB_AID_LINE_ITEM	System assigned invoice number which uniquely identifies each off line order invoice.	NUMBER	NOT NULL
InvoiceNumber_ID	PROGRAM_LINE_ITEM	System assigned invoice number which uniquely identifies each off line order invoice.	NUMBER	NOT NULL
InvoiceNumber_ID	TRNG_NCO_LINE_ITEM	System assigned invoice number which uniquely identifies each off line order invoice.	NUMBER	NOT NULL
ItemDescription	TRNG_NCO_MATERIAL	Description of the Training NCO Item	VARCHAR2(200)	NULL
ItemName	TRNG_NCO_MATERIAL	Name of the Training NCO Item	VARCHAR2(65)	NOT NULL
ItemQty	TRNG_NCO_LINE_ITEM	Quantity of the item ordered	NUMBER	NOT NULL
Item_ID	TRNG_NCO_LINE_ITEM	Identifies the Training NCO Item	VARCHAR2(6)	NOT NULL
Item_ID	TRNG_NCO_MATERIAL	Identifies the Training NCO Item	VARCHAR2(6)	NOT NULL
JACompQuantity	JOB_AID_COMPONENTS_X	Quantity of the component to be included in the job aid	NUMBER	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
JAQuantity	JOB_AID_LINE_ITEM	Quantity of this job aid requested.	NUMBER	NOT NULL
JobAidDescription	JOB_AID	Narrative description of each item	VARCHAR2(600)	NOT NULL
JobAidName	JOB_AID	Name associated with each job aid as provided by the LOGAIS system.	VARCHAR2(30)	NOT NULL
JobAidOnHandQty	JOB_AID	Quantity of job aids on hand, as provided by LOGAIS update.	NUMBER	NOT NULL
JobAid_ID	JOB_AID	Unique ID associated with every job aid. Ideally, these values will be provided and updated via interface with LOGAIS inventory software.	VARCHAR2(10)	NOT NULL
JobAid_ID	JOB_AID_COMPONENTS_X	Unique ID associated with every job aid. Ideally, these values will be provided and updated via interface with LOGAIS inventory software.	VARCHAR2(10)	NOT NULL
JobAid_ID	JOB_AID_LINE_ITEM	Unique ID associated with every job aid. Ideally, these values will be provided and updated via interface with LOGAIS inventory software.	VARCHAR2(10)	NOT NULL
JrCrs_ID	COURSE_PREREQ	Course number with version number if	VARCHAR2(6)	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		applicable		
JrProg_ID	PROG_PREREQ	Program number with version number if applicable	VARCHAR2(6)	NOT NULL
MCTFAddr1	MCTF_PERS	First address line of the Home Address for a Marine contained in the MCTFS database. Used as the mailing Address for Reservists	VARCHAR2(46)	NULL
MCTFAddr2	MCTF_PERS	Second Address line of the Home Address for a Marine contained in the MCTFS database. Used as the mailing Address for Reservists	VARCHAR2(46)	NULL
MCTFBilletMOS	MCTF_PERS	Billet MOS of a Marine contained in the MCTF database	CHAR(4)	NULL
MCTFCity	MCTF_PERS	City of the Home Address for a Marine contained in the MCTFS database. Used as the mailing Address for Reservists	VARCHAR2(34)	NULL
MCTFComponent	MCTF_PERS	Component of a Marine contained in the MCTFS database.	CHAR(2)	NOT NULL
MCTFFirstInitial	MCTF_PERS	First Initial a Marine contained in the MCTFS database.	VARCHAR2(1)	NOT NULL
MCTFGrade	MCTF_PERS	Grade of a Marine contained in the	CHAR(2)	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		MCTFS database.		
MCTFLastName	MCTF_PERS	Last Name of a Marine contained in the MCTFS database.	VARCHAR2(20)	NOT NULL
MCTFMOS	MCTF_PERS	Primary MOS of a Marine contained in the MCTFS database.	CHAR(4)	NOT NULL
MCTFMiddleInitial	MCTF_PERS	Middle Initial of a Marine contained in the MCTFS database.	VARCHAR2(1)	NULL
MCTFPlatoonCode	MCTF_PERS	Foreign Key which identifies the Platoon Code of a Marine contained in the MCTFS database.	CHAR(4)	NOT NULL
MCTFRank	MCTF_PERS	Rank of a Marine contained in the MCTFS database. This is a calculated value. The procedure used to calculate it compares the Grade and MOS to determine the Rank.	VARCHAR2(6)	NOT NULL
MCTFSSN_ID	MCTF_PERS	Social Security Number which uniquely identifies the instance of a Marine contained in the MCTFS database.	CHAR(9)	NOT NULL
MCTFSelToGrade	MCTF_PERS	Grade to which selected of a Marine contained in the MCTFS database.	CHAR(2)	NULL
MCTFZipCode	MCTF_PERS	Zip Code of the Home Address for a Marine	VARCHAR2(9)	NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		contained in the MCTFS database. Used as the mailing Address for Reservists		
MCTF_ECC	MCTF_PERS	End of Current Contract of a Marine contained in the MCTFS database.	DATE	NULL
MCTF_RUCMCC	MCTF_PERS	Reporting Unit Code and Monitored Command Code from MCTF	VARCHAR2(8)	NOT NULL
NMSAddr1	NON_MARINE_STUDENT_ADDRESS	First line of student's address, for non-marine students	VARCHAR2(46)	NOT NULL
NMSAddr2	NON_MARINE_STUDENT_ADDRESS	Second line of student's address, for non-marine students	VARCHAR2(46)	NULL
NMSCity	NON_MARINE_STUDENT_ADDRESS	City of student's address, for non-marine students	VARCHAR2(36)	NULL
NMSZipCode	NON_MARINE_STUDENT_ADDRESS	Zip code of student's address, for non-marine students	VARCHAR2(9)	NOT NULL
OpscanNumber	ERROR_LISTING	The number assigned to the error listing by the SCANTRON reader.	VARCHAR2(4)	NOT NULL
P_Accredited	PROGRAM	Flag which indicates if a program is accredited	CHAR(1)	NULL
P_GradeElOther	PROGRAM	Prerequisite grade of non-USMC student's wishing to enroll in this program.	CHAR(2)	NULL
P_GradeElUSMC	PROGRAM	Prerequisite USMC	CHAR(2)	NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		grade associated with this program. USMC students must be of this grade or selected to this grade in order to enroll.		
Prerequisites	PROGRAM	course prerequisites for the program	VARCHAR2(500)	NULL
PrgCCD	STUDENT_PROGRAM_X	Date this instance of an enrollment is to be scrubbed from the data base.	DATE	NULL
PrgCompQty	PROG_COMPONENT_X	quantity of this component to be associated with this program	NUMBER	NOT NULL
PrgCreditHours	PROGRAM	Credit hours associated with the program. Must contain a value if program is accredited. Used for transcript purposes.	VARCHAR2(3)	NULL
PrgDateClosed	PROGRAM	Date program closed for enrollment	DATE	NULL
PrgDateOpened	PROGRAM	Date program opened for enrollment	DATE	NULL
PrgDiplomaIssueDate	STUDENT_PROGRAM_X	Date diploma for program completion is issued to the student.	DATE	NULL
PrgEnrollDate	STUDENT_PROGRAM_X	Date student was enrolled in the program	DATE	NOT NULL
PrgOnHandQty	PROGRAM	Quantity currently on hand, as provided by interface with inventory	NUMBER	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		table		
PrgQuantity	PROGRAM_LINE_ITEM	Quantity of this program requested.	NUMBER	NOT NULL
PrgStudyHours	PROGRAM	Study Hours associated with program. Used for transcript generation purposes.	VARCHAR2(3)	NULL
PrgTitle	PROGRAM	Title of the program, as defined in the course catalog	VARCHAR2(60)	NOT NULL
ProgramAbbreviation	PROGRAM	Program Abbreviation as used for MMS updates	VARCHAR2(20)	NOT NULL
Program_ID	COURSE_PROGRAM_X	Program number with version number if applicable	VARCHAR2(6)	NOT NULL
Program_ID	PROGRAM	Program number with version number if applicable	VARCHAR2(6)	NOT NULL
Program_ID	PROGRAM_LINE_ITEM	Program number with version number if applicable	VARCHAR2(6)	NOT NULL
Program_ID	PROG_COMPONENT_X	Program number with version number if applicable	VARCHAR2(6)	NOT NULL
Program_ID	STUDENT_PROGRAM_X	Program number with version number if applicable	VARCHAR2(6)	NOT NULL
Program_ID	STUD_PRG_TRAN_X	Program number with version number if applicable	VARCHAR2(6)	NOT NULL
Purpose	PROGRAM	Purpose of Program	VARCHAR2(2000)	NULL
QtyOnHand	TRNG_NCO_MATERIAL	Quantity of item on	NUMBER	NOT

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		hand in warehouse		NULL
Quantity	COURSE_COMPONENTS_X	Quantity of the component to be included in the course	NUMBER	NOT NULL
QuestionNumber	MULTIPLE_CHOICE_EXAM	Unique number which identifies the number of the question on this specific Exam instance.	NUMBER	NOT NULL
RUCAddr1	REPORTING_UNIT_CODE	First address line of unit address	VARCHAR2(46)	NOT NULL
RUCAddr2	REPORTING_UNIT_CODE	Second address line from unit address	VARCHAR2(46)	NULL
RUCCityState	REPORTING_UNIT_CODE	String representing the City and State associated with each unit address.	VARCHAR2(36)	NOT NULL
RUCMCC	REPORTING_UNIT_CODE	Reporting Unit Code and Monitored Command Code from MCTF	VARCHAR2(8)	NOT NULL
RUCZip	REPORTING_UNIT_CODE	Zip code associated with each unit address.	VARCHAR2(9)	NOT NULL
RankDescription	RANK	Description of the rank abbreviation contained in the primary key. Describes the rank and its service affiliation	VARCHAR2(45)	NOT NULL
Rank_ID	RANK	Abbreviation normally associated with a specific rank for a specific military service or services.	VARCHAR2(6)	NOT NULL
Rank_ID	STUDENT	Abbreviation normally	VARCHAR2(6)	NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		associated with a specific rank for a specific military service or services.		
RemMethDescription	REMEDIATION_METHOD	Description of each remediation method.	VARCHAR2(30)	NOT NULL
RemediationMeth	COURSE	Method for remediation associated with each course or program	VARCHAR2(1)	NOT NULL
RemediationMeth	PROGRAM	Method for remediation associated with each course or program	VARCHAR2(1)	NOT NULL
RemediationMeth	REMEDIATION_METHOD	Method for remediation associated with each course or program	VARCHAR2(1)	NOT NULL
SC_Comp	STUDENT_COURSE_X	Flag which indicates that this course has been completed by the student	CHAR(1)	NOT NULL
SC_PME	STUDENT_COURSE_X	Flag which indicates if this enrollment is part of a PME program enrollment	CHAR(1)	NOT NULL
SC_TransDate_ID	STUD_CRS_TRAN_X	Date of this transaction instance.	DATE	NOT NULL
SP_Comp	STUDENT_PROGRAM_X	Flag which indicates the student has completed this program.	CHAR(1)	NOT NULL
SP_TransDate_ID	STUD_PRG_TRAN_X	Date of this transaction instance.	DATE	NOT NULL
SchoolCode	PROGRAM	School code associated with the program. Used for MMS purposes.	CHAR(3)	NOT NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		Relates program to a specific resident school (e.g., AWS, etc.)		
ServCompDescription	SERVICE_COMPONENT	Description of service component for corresponding Component_ID	VARCHAR2(40)	NOT NULL
ServiceComponent_ID	SERVICE_COMPONENT	Alpha-numeric code for service component. Defined by corresponding Description.	CHAR(2)	NOT NULL
ServiceComponent_ID	STUDENT	Alpha-numeric code for service component. Defined by corresponding Description.	CHAR(2)	NOT NULL
SrCrs_ID	COURSE_PREREQ	Course number with version number if applicable	VARCHAR2(6)	NOT NULL
SrProg__ID	PROG_PREREQ	Program number with version number if applicable	VARCHAR2(6)	NOT NULL
State	MCTF_PERS	State of the Home Address for a Marine contained in the MCTFS database. Used as the mailing Address for Reservists	VARCHAR2(2)	NULL
StateDescription	STATE	Full state name associated with each state.	VARCHAR2(20)	NOT NULL
State_ID	CUSTOMER	Standard two letter state	CHAR(2)	NULL

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		abbreviation		
State_ID	NON_MARINE_STUDENT_ADDRESS	Standard two letter state abbreviation	CHAR(2)	NULL
State_ID	STATE	Standard two letter state abbreviation	CHAR(2)	NOT NULL
StatusCode	COURSE	Status Code for course status	VARCHAR2(2)	NOT NULL
StatusCode	PROGRAM	Status Code for course status	VARCHAR2(2)	NOT NULL
StatusCodeDescription	STATUS_CODE	Definition of status code for course status	VARCHAR2(60)	NOT NULL
StatusCode_ID	STATUS_CODE	Status Code for course status	VARCHAR2(2)	NOT NULL
StudFirstName	STUDENT	Student's first name	VARCHAR2(10)	NOT NULL
StudLastName	STUDENT	Student's last name	VARCHAR2(20)	NOT NULL
StudMI	STUDENT	Student's middle initial	VARCHAR2(1)	NULL
StudentSSN_ID	ERROR_LISTING	Social security number of student	VARCHAR2(9)	NULL
StudentSSN_ID	NON_MARINE_STUDENT_ADDRESS	Social security number of student	VARCHAR2(9)	NOT NULL
StudentSSN_ID	STUDENT	Social security number of student	VARCHAR2(9)	NOT NULL
StudentSSN_ID	STUDENT_ANSWERS	Social security number of student	VARCHAR2(9)	NOT NULL
StudentSSN_ID	STUDENT_COURSE_EXAM_X	Social security number of student	VARCHAR2(9)	NOT NULL
StudentSSN_ID	STUDENT_COURSE_X	Social security number of student	VARCHAR2(9)	NOT NULL
StudentSSN_ID	STUDENT_PROGRAM_X	Social security number of student	VARCHAR2(9)	NOT NULL
StudentSSN_ID	STUD_CRS_TRAN_X	Social security number	VARCHAR2(9)	NOT

Attribute Name	Entity Header	Attribute Definition	Column Datatype	Null Option
		of student		NULL
StudentSSN_ID	STUD_PRG_TRAN_X	Social security number of student	VARCHAR2(9)	NOT NULL
TransactionCodeDescription	TRANSACTION_CODES	Description of each transaction code.	VARCHAR2(200)	NOT NULL
Transaction_ID	STUD_CRS_TRAN_X	Code identifying type of transaction	CHAR(1)	NOT NULL
Transaction_ID	STUD_PRG_TRAN_X	Code identifying type of transaction	CHAR(1)	NOT NULL
Transaction_ID	TRANSACTION_CODES	Code identifying type of transaction	CHAR(1)	NOT NULL
UnitName	REPORTING_UNIT_CODE	English descriptive name of each unit by RUCMCC	VARCHAR2(45)	NOT NULL

APPENDIX O. SSD AND MIS LEVEL RELATIONSHIP DEFINITIONS

Physical Name	Parent Entity	Verb Phrase	Child Entity	Relationship Type	Relationship Definition	Nulls	Cardinality
Belongs_to	EXAM	Belongs to	STUDENT_COURSE_EXAM_X	Identifying	An exam belongs to zero, one, or many students.		One-to-Zero-One-or-More
Contains	COURSE	Contains	STUDENT_COURSE_X	Identifying	A course contains enrolled students. A course may contain zero, one, or many enrolled students.		One-to-Zero-One-or-More
Defines_Grading_Method	CATEGORY_CODE	Defines Grading Method	EXAM	Non-identifying	The category code defines the grading method for an exam. A category code can belong to zero, one or many exams, and an exam must have one and only one category code.	No Nulls	One-to-Zero-One-or-More
Defines_Service_Affiliation_of	SERVICE_COMPONENT	Defines Service Affiliation of	STUDENT	Non-identifying	The component defines the service affiliation of a student. A component can belong to zero, one, or many students, and a student must have one and only one component. Civilians also have a component code.	No Nulls	One-to-Zero-One-or-More
Eval_By	COURSE	Evaluated-By	EXAM	Identifying	A course has zero, one, or more associated exams. A course not yet opened or already closed may not have any		One-to-Zero-One-or-More

Physical Name	Parent Entity	Verb Phrase	Child Entity	Relationship Type	Relationship Definition	Nulls	Cardinality
					exams associated with it. An exam belongs to one and only one course.		
Is_contained_on	ERROR_CODES	Is contained on	ERROR_LISTING	Non-identifying	An error code is contained on zero, one or many error listings. An error listing contains one and only one error code.	No Nulls	One-to-Zero-One-or-More
Is_enrolled_in	STUDENT	Is enrolled in	STUDENT_COURSE_X	Identifying	A student is enrolled in one or more courses.		One-to-Zero-One-or-More
Is_enrolled_in	STUDENT	Is enrolled in	STUDENT_PROGRAM_X	Identifying	A student is enrolled in zero, one, or many programs.		One-to-Zero-One-or-More
Is_issued_a	STUDENT	Is issued a	STUDENT_COURSE_EXAM_X	Identifying	A student is issued zero, one, or many exams.		One-to-Zero-One-or-More
Is_ordered_as	COMPONENTS	Is ordered as	COMPONENT_LINE_ITEM	Identifying	A course component ordered off line belongs to a course component line item on an invoice. A course component line item must contain one and only one course component, but a course component may be part of zero, one, or many line items, so long as		One-to-Zero-One-or-More

Physical Name	Parent Entity	Verb Phrase	Child Entity	Relationship Type	Relationship Definition	Nulls	Cardinality
					it appears at most once on a particular invoice.		
Is_ordered_as	COURSE	Is ordered as	COURSE_LINE_ITEM	Identifying	A course ordered off line belongs to a course line item on an invoice. A course line item must contain one and only one course, but a course may be part of zero, one, or many line items, so long as it appears at most once on a particular invoice.		One-to-Zero-One-or-More
Is_ordered_as	PROGRAM	Is ordered as	PROGRAM_LINE_ITEM	Identifying	A program ordered off line belongs to a program line item on an invoice. A program line item must contain one and only one program, but a program may be part of zero, one, or many line items, so long as it appears at most once on a particular invoice.		One-to-Zero-One-or-More
associated_with	TRANSACTION_CODES	associated with	STUD_CRS_TRAN_X	Identifying	A transaction code is associated with zero, one, or many student		One-to-Zero-One-or-More

Physical Name	Parent Entity	Verb Phrase	Child Entity	Relationship Type	Relationship Definition	Nulls	Cardinality
					course transactions		
associated_with	TRANSACTION_CODES	associated with	STUD_PRG_TRAN_X	Identifying	A transaction code is associated with zero, one, or many student program transactions		One-to-Zero-One-or-More
comprises_a	COMPONENTS	comprises a	COURSE_COMPONENTS_X	Identifying	A course component belongs to zero one or many courses. For example, a component may belong only to job aids, not to a course.		One-to-Zero-One-or-More
comprises_a	COMPONENTS	comprises a	EXAM_COMPONENT_X	Identifying	course components comprise exams		One-to-Zero-One-or-More
comprises_a	COMPONENTS	comprises a	JOB_AID_COMPONENTS_X	Identifying	A course component belongs to zero one or many job aids. For example, a job aid may not belong to any courses.		One-to-Zero-One-or-More
contains	INVOICE	contains	COMPONENT_LINE_ITEM	Identifying	An off line request invoice contains zero, one, or many course component line items. A course component line item belongs to one and only one invoice.		One-to-Zero-One-or-More
contains	INVOICE	contains	COURSE_LINE_ITEM	Identifying	An off line		One-to-Zero-

Physical Name	Parent Entity	Verb Phrase	Child Entity	Relationship Type	Relationshlp Definition	Nulls	Cardinality
					request invoice contains zero, one, or many course line items. A course line item belongs to one and only one invoice.		One-or-More
contains	INVOICE	contains	JOB_AID_LINE_ITEM	Identifying	An off line request invoice contains zero, one, or many job aid line items. A job aid line item belongs to one and only one invoice.		One-to-Zero-One-or-More
contains	INVOICE	contains	PROGRAM_LINE_ITEM	Identifying	An off line request invoice contains zero, one, or many program line items. A program line item belongs to one and only one invoice.		One-to-Zero-One-or-More
contains	INVOICE	contains	TRNG_NCO_LINE_ITEM	Identifying	An off line request invoice contains zero, one, or many training NCO material items. A training NCO material item belongs to one and only one invoice		One-to-Zero-One-or-More
contains	PROGRAM	Contains	STUDENT_PROGRAM_X	Identifying	A program contains enrolled students. A program may		One-to-Zero-One-or-More

Physical Name	Parent Entity	Verb Phrase	Child Entity	Relationship Type	Relationship Definition	Nulls	Cardinality
					contain zero, one, or many enrolled students.		
contains	STUDENT_COURSE_EXAM_X	contains	ERROR_LISTING	Non-identifying	A student_course_exam instance contains zero, one, or many error listings generated by the Scantron reader. An error listing belongs to zero or one student_course_exam instance.	Nulls Allowed	Zero-or-One-to-Zero-One-or-More
defines_State_of	STATE	defines State of	CUSTOMER	Non-identifying	A state belongs to zero, one, or many customer address records. A customer instance has one state associated with it.	Nulls Allowed	Zero-or-One-to-Zero-One-or-More
defines_grade_of	GRADE	defines grade of	STUDENT	Non-identifying	defines the grade of a student	Nulls Allowed	Zero-or-One-to-Zero-One-or-More
defines_rank_of	RANK	defines rank of	STUDENT	Non-identifying	defines the rank of a student	Nulls Allowed	Zero-or-One-to-Zero-One-or-More
defines_state_of	STATE	defines state of	NON_MARINE_STUDENT_ADDRESS	Non-identifying		Nulls Allowed	Zero-or-One-to-Zero-One-or-More
defines_status_of	STATUS_CODE	Defines Status Of	PROGRAM	Non-identifying	The Status code defines the status of zero, one, or many programs. A program has one and only one status.	No Nulls	One-to-Zero-One-or-More
defines_status_of	STATUS_CODE	Defines status of	COURSE	Non-identifying	The status code defines the status	No Nulls	One-to-Zero-One-or-More

Physical Name	Parent Entity	Verb Phrase	Child Entity	Relationship Type	Relationship Definition	Nulls	Cardinality
					of zero, one, or many courses. A course has one and only one status code.		
has_a	ANSWER_CODE	has a	MULTIPLE_CHOICE_EXAM	Non-identifying	An answer code is associated with a multiple choice exam question. It defines what is an acceptable answer for the specified question.	No Nulls	One-to-Zero-One-or-More
has_a_prereq_of	COURSE	has a prereq of	COURSE_PREREQ	Identifying	A course can have a prerequisite of zero, one, or many other courses		One-to-Zero-One-or-More
has_an_associated	STUDENT_COURSE_X	has an associated	STUD_CRS_TRAN_X	Identifying	A student_course enrollment has zero, one, or many associated transactions.		One-to-Zero-One-or-More
has_an_associated	STUDENT_PROGRAM_X	has an associated	STUD_PRG_TRAN_X	Identifying	A student_program enrollment has zero, one, or many associated transactions.		One-to-Zero-One-or-More
has_assoc	REPORTING_UNIT_CODE	has associated	MCTF_PERS	Non-identifying	A RUCMCC has zero, one, or many associated MCTF personnel. A MCTF personnel instance belongs to zero or one RUCMCC	No Nulls	One-to-Zero-One-or-More
have_associated	ERROR_LISTING	have associated	ERR_ANS_STRING	Identifying	Error listings have associated		One-to-Zero-One-or-More

Physical Name	Parent Entity	Verb Phrase	Child Entity	Relationship Type	Relationship Definition	Nulls	Cardinality
					exam answer strings		
have_associated	STUDENT_COURSE_EXAM_X	have associated	STUDENT_ANSWERS	Identifying	Exams received from student's have associated answers stored in the database, if the exams have been machine graded.		One-to-Zero-One-or-More
is_a_prereq_for	COURSE	is a prereq for	COURSE_PREREQ	Identifying	A course can be the prerequisite for zero, one, or many other courses		One-to-Zero-One-or-More
is_assoc_with	ED_LEVEL	Is associated with	PROGRAM	Non-identifying	An education level is associated with zero, one, or many programs. A program has zero or one associated education levels.	Nulls Allowed	Zero-or-One-to-Zero-One-or-More
is_assoc_with	ED_LEVEL	is associated with	COURSE	Non-identifying	An education level is associated with zero, one, or many courses. A course has zero or one associated education levels.	Nulls Allowed	Zero-or-One-to-Zero-One-or-More
is_associated_with	CUSTOMER	is associated with	INVOICE	Non-identifying	a customer is associated with an order invoice	No Nulls	One-to-One-or-More (P)
is_associated_with	REMEDIATION_METHOD	is associated with	COURSE	Non-identifying	A remediation method is associated with zero, one or many courses. A course has one and only one remediation method	No Nulls	One-to-Zero-One-or-More

Physical Name	Parent Entity	Verb Phrase	Child Entity	Relationship Type	Relationship Definition	Nulls	Cardinality
is_associated_with	REMEDIATION_METHOD	is associated with	PROGRAM	Non-identifying	A remediation method is associated with zero, one or many courses. A course has one and only one remediation method	No Nulls	One-to-Zero-One-or-More
is_comprised_of	COURSE	Is comprised of	COURSE_COMPONENTS_X	Identifying	A course is comprised of zero, one or many components. When a course is first created, it may have no physical components associated with it		One-to-Zero-One-or-More
is_comprised_of	EXAM	is comprised of	EXAM_COMPONENT_X	Identifying	exams are comprised of course components		One-to-Zero-One-or-More
is_comprised_of	JOB_AID	is comprised of	JOB_AID_COMPONENTS_X	Identifying	A job aid is comprised of zero, one or many components. When a job aid is first created, it may have no physical components associated with it		One-to-Zero-One-or-More
is_made_up_of	PROGRAM	is made up of	COURSE_PROGRAM_X	Identifying	A program is made up of zero, one, or more courses.		One-to-Zero-One-or-More
is_ordered_as	JOB_AID	is ordered as	JOB_AID_LINE_ITEM	Identifying	A job aid ordered off line belongs to a job aid line item on an invoice. A job		One-to-Zero-One-or-More

Physical Name	Parent Entity	Verb Phrase	Child Entity	Relationship Type	Relationship Definition	Nulls	Cardinality
					aid line item must contain one and only one job aid, but a job aid may be part of zero, one, or many line items, so long as it appears at most once on a particular invoice.		
is_ordered_as	TRNG_NCO_MATERIAL	is ordered as	TRNG_NCO_LINE_ITEM	Identifying	A training NCO material item ordered off line belongs to a training NCO material item line item on an invoice. A training NCO material item line item must contain one and only one training NCO material item, but a training NCO material item may be part of zero, one, or many line items, so long as it appears at most once on a particular invoice.		One-to-Zero-One-or-More
is_part_of	COURSE	is part of	COURSE_PROGRAM_X	Identifying	A course may be a part of zero, one, or many programs.		One-to-Zero-One-or-More
may_be_included_in	COMPONENTS	may be included in	PROG_COMPONENT_X	Identifying	A component may be included		One-to-Zero-One-or-More

Physical Name	Parent Entity	Verb Phrase	Child Entity	Relationship Type	Relationship Definition	Nulls	Cardinality
					in a program directly, without being associated directly with the programs sub-courses.		
may_consist_of	PROGRAM	may consist of	PROG_COMPONENT_X	Identifying	A program may have associated components, other than those components associated with the program's included courses		One-to-Zero-One-or-More
may_have_an	STUDENT	may have an	NON_MARINE_STUDENT_ADDRESS	Identifying	a non marine student has an address		One-to-Zero-or-One (Z)
opens_an	SSD_CLERK	opens an	INVOICE	Non-identifying	an SSD clerk is responsible for opening an invoice	Nulls Allowed	Zero-or-One-to-Zero-One-or-More
prereq_for	PROGRAM	Is a prerequisite for	PROG_PREREQ	Identifying	A program can be the prerequisite for zero, one, or many other programs		One-to-Zero-One-or-More
prereq_of	PROGRAM	Has a prerequisite of	PROG_PREREQ	Identifying	A program can have a prerequisite of zero, one, or many other programs		One-to-Zero-One-or-More
	EXAM	is a	MULTIPLE_CHOICE_EXAM	Subtype	An exam may be of subtype multiple choice exam		Is a

APPENDIX P. SSD AND MIS LEVEL DATA MODEL

(See pocket on inside back cover.)

APPENDIX Q. STEPS REQUIRED TO AVOID THE ORACLE MUTATING TABLE ERROR WHEN USING ER*WIN*

The ORATRG.ER1 model that ships with ER*win* contains trigger templates that avoid the mutating table error common to ORACLE 7 triggers.

To use these templates with your existing model, utilize the following steps:
1. Open the ORATRG.ER1 model and save it under another name (e.g., ORATRG1.ER1).
2. Open the Server, ORACLE Trigger Template Editor drop down from the menu bar.
3. Select ORACLE table package from the User Override window.
4. Copy the template code to the Windows clipboard.
5. Close the ORACLE Trigger Template Editor.
6. Right click on the CUSTOMER entity and select ORACLE Table Property, Pre & Post Script.
7. Click the Script Template editor button.
8. Click the new button and name the new script template "ORACLE table package".
9. Paste the contents of the Windows clipboard into the Table Script Template window.
10. Check the attach to new entity box for the ORACLE non-mutating NEW table package, ORACLE non-mutating OLD table package, and ORACLE table package script templates.
11. Click OK.
12. Click OK.
13. Delete all the entities except CUSTOMER and all the text blocks from this model.
14. Edit/Copy Model.
15. Edit/Paste into your existing model. This will copy trigger templates from this model into your existing model.
16. Delete the CUSTOMER entity.
17. Open Oracle Trigger Template Editor.
18. Attach trigger templates in your existing model as they are attached in this model. (e.g. CHILD INSERT RESTRICT == ORACLE child insert restrict).
19. Attach ALL EXISTING ENTITIES to the three entity pre-scripts using the Table Property/Pre and Post Script Editor. Step 10 above will assure that the three entity pre-scripts are attached to ALL NEW ENTITIES.
20. Regenerate your model with the same schema generation options as in ORATRG.1 model. Make sure declarative referential integrity is off, triggers are on, and Table Pre-Script is on.

Comments:
1. These templates will not work correctly with recursive relationships.
2. You should not mix these templates with declarative referential integrity in the same relationship path. See comments on page 1 of the ORATRG.ERQ1 model.
3. Actions in the trigger template editor that do not have templates attached are unsupported.

APPENDIX R. SERVER SIDE TRIGGER PL/SQL SOURCE CODE

I. Student course enrollment transaction codes: The following set of triggers, procedures, and functions allow the appropriate transaction code to be automatically entered in the student course enrollment transaction log (i.e., STUD_CRS_TRAN_X table) when a student is enrolled in a course (i.e., an instance is inserted into the STUD_CRS_X table).

```
/*
```

This trigger allows the PL/SQL table to be initialized (emptied) and the corresponding new values which are being entered into the STUD_CRS_X table to be read into the PL/SQL table.

Trigger Name: TIROW_STUD_CRS_TRAN_X
Associated Table: STUD_CRS_X
Type Trigger: For Each Row, After Insert

Source Code: */

```
BEGIN
STUD_CRS_X_TRAN.initialize_table;
if (STUD_CRS_X_TRAN.get_nest_level = 0) then
   STUD_CRS_X_TRAN.insert_tran(
   :new.Crs_ID,
   :new.StudSSN_ID,
   :new.CrsEnrollDate);
 end if;
END;
```

```
/*
```

This trigger reads each row of the PL/SQL table populated with new STUD_CRS_X enrollments into a temporary record; it then inserts these values from the temporary record into the STUD_CRS_TRAN_X table, along with the appropriate transaction code (i.e., "A").

Trigger Name: TI_STUD_CRS_TRAN_X
Associated Table: STUD_CRS_X
Type Trigger: After Insert

Source Code */

```
declare
 numrows INTEGER;
 rowindex INTEGER;
 trannewrec STUD_CRS_X_TRAN.STUD_CRS_X_TRAN;
 newrec STUD_CRS_X_TRAN.STUD_CRS_X_row;
begin

  if (STUD_CRS_X_TRAN.get_nest_level = 0) then
   STUD_CRS_X_TRAN.increment_nest_level;
   rowindex := 1;
   while rowindex <= STUD_CRS_X_TRAN.get_row_count loop
    STUD_CRS_X_TRAN.TRAN_by_index(rowindex,trannewrec);
    select * into newrec from STUD_CRS_X
    where
     trannewrec.Crs_ID = STUD_CRS_X.Crs_ID and
     trannewrec.StudSSN_ID = STUD_CRS_X.StudSSN_ID and
```

```
      trannewrec.CrsEnrollDate = STUD_CRS_X.CrsEnrollDate;

 INSERT INTO STUD_CRS_TRAN_X
    VALUES (newrec.crs_id, newrec.StudSSN_ID, 'A', newrec.crsenrolldate);

rowindex := rowindex + 1;
   end loop;
   STUD_CRS_X_TRAN.decrement_nest_level;
  end if;
end;

/*
```

This package specification contains the specification for a series of functions and procedures associated with the two triggers described above. This package allows for the insertion of transaction codes into the STUD_CRS_TRAN_X table upon enrollment of new students.

Package Specification for Package: STUD_CRS_X_TRAN

Source Code */

```
PACKAGE    STUD_CRS_X_TRAN AS
/*
*****************************************
*****************************************
Package SPECIFICATION  STUD_CRS_X_TRAN for creating PL/SQL tables for use
in updating transaction records to avoid mutating tables
By Aaron Slaughter
28 April 97
*****************************************
*****************************************
*/
/* used to return rows from PL/SQL tables */
 type STUD_CRS_X_TRAN is record
      (Crs_ID VARCHAR2(6),
       StudSSN_ID VARCHAR2(9),
       CrsEnrollDate DATE);

 /* used to access rows in trigger templates */
 type STUD_CRS_X_row is record
      (Crs_ID VARCHAR2(6),
       StudSSN_ID VARCHAR2(9),
       CrsEnrollDate DATE,
       CrsCCD DATE,
       CrsFinalGrade VARCHAR2(3),
       CrCertDate DATE,
       SC_PME CHAR(1),
       SC_Comp CHAR(1),
       Disenrolled CHAR(1));

 /* used in recursive relationship situations */
 function get_nest_level return number;
 procedure increment_nest_level;
 procedure decrement_nest_level;
```

```
  /* resets PL/SQL table contents */
  procedure initialize_table;

  /* inserts a row in to PL/SQL table */
  procedure INSERT_TRAN(
    Crs_ID IN VARCHAR2,
    StudSSN_ID IN VARCHAR2,
    CrsEnrollDate IN DATE
  );

  /* determines if there are any more rows in PL/SQL tables */
  function more_rows return boolean;

  /* determines the number of rows in the PL/SQL table */
  function get_row_count return number;

  /* gets the next row and decrements row count thereby deleting row
     in PL/SQL tables */
  procedure next_TRAN (tbl_row out STUD_CRS_X_TRAN);

  /* gets row by index from PL/SQL tables */
  procedure TRAN_by_index(n in number, tbl_row out STUD_CRS_X_TRAN);

end;

/*
```

This package body contains the source code for a series of functions and procedures associated with the two triggers described above. This package allows for the insertion of transaction codes into the STUD_CRS_TRAN_X table upon enrollment of new students.

Package Body for Package: STUD_CRS_X_TRAN

Source Code */

```
PACKAGE BODY    STUD_CRS_X_TRAN IS
/*
*****************************************
*****************************************
Package BODY STUD_CRS_X_TRAN for creating PL/SQL tables for use
in updating transaction records to avoid mutating tables
By Aaron Slaughter
28 April 97
*****************************************
*****************************************
*/
/* declare PL/SQL table types to store VALUES       from STUD_CRS_X for INSERT into
STUD_CRS_TRANS_X */
  type Crs_ID_typ is table of
    VARCHAR2(6) index by binary_integer;
  type StudSSN_ID_typ is table of
    VARCHAR2(9) index by binary_integer;
  type CrsEnrollDate_typ is table of
    DATE index by binary_integer;
```

```
 /* declare PL/SQL tables to store VALUES from STUD_CRS_X for INSERT into
STUD_CRS_TRANS_X */
 Crs_ID_tbl Crs_ID_typ;
 StudSSN_ID_tbl StudSSN_ID_typ;
 CrsEnrollDate_tbl CrsEnrollDate_typ;


 /* row count in PL/SQL tables */
 nrows number;

 /* nest level */
 TriggerNestLevel number := 0;

 procedure initialize_table is
 begin
  nrows := 0;
  TriggerNestLevel := 0;
 end;

 function get_nest_level return number is
 begin
  return TriggerNestLevel;
 end;

 procedure increment_nest_level is
 begin
  TriggerNestLevel := TriggerNestLevel + 1;
 end;

 procedure decrement_nest_level is
 begin
  TriggerNestLevel := TriggerNestLevel - 1;
 end;

 procedure INSERT_TRAN(
  Crs_ID IN VARCHAR2,
  StudSSN_ID IN VARCHAR2,
  CrsEnrollDate IN DATE
 ) is
 begin
  nrows := nrows + 1;
  Crs_ID_tbl(nrows) := Crs_ID;
  StudSSN_ID_tbl(nrows) := StudSSN_ID;
  CrsEnrollDate_tbl (nrows) := CrsEnrollDate;


 end;

 function more_rows return boolean is
 begin
  return (nrows > 0);
 end;

 function get_row_count return number is
 begin
```

```
  return (nrows);
 end;

 procedure next_TRAN (tbl_row out STUD_CRS_X_TRAN) is
 begin
  tbl_row.Crs_ID := Crs_ID_tbl(nrows);
  tbl_row.StudSSN_ID := StudSSN_ID_tbl(nrows);
  tbl_row.CrsEnrollDate := CrsEnrollDate_Tbl (nrows);

  nrows := nrows - 1;
 end;

 procedure TRAN_by_index (n in number, tbl_row out STUD_CRS_X_TRAN) is
 begin
  if n > 0 and n <= nrows then
   tbl_row.Crs_ID := Crs_ID_tbl(n);
   tbl_row.StudSSN_ID := StudSSN_ID_tbl(n);
   tbl_row.CrsEnrollDate := CrsEnrollDate_tbl(n);

  end if;
 end;

end;
```

II. Student program enrollment transaction codes: The following set of triggers, procedures, and functions allow the appropriate transaction code to be automatically entered in the student program enrollment transaction log (i.e., STUD_PRG_TRAN_X table) when a student is enrolled in a program (i.e., an instance is inserted into the STUD_PRG_X table).

```
/*
```

This trigger allows the PL/SQL table to be initialized (emptied) and the corresponding new values which are being entered into the STUD_PRG_X table to be read into the PL/SQL table.

```
Trigger Name: TIROW_STUD_PRG_TRAN_X
Associated Table:  STUD_PRG_X
Type Trigger:  For Each Row,  After Insert



Source Code: */

BEGIN
STUD_PRG_X_TRAN.initialize_table;
if (STUD_PRG_X_TRAN.get_nest_level = 0) then
   STUD_PRG_X_TRAN.insert_tran(
   :new.Prg_ID,
   :new.StudSSN_ID,
   :new.PrgEnrollDate);
 end if;
END;

/*
```

This trigger reads each row of the PL/SQL table populated with new STUD_PRG_X enrollments into a temporary record; it then inserts these values from the temporary record into the STUD_PRG_TRAN_X table, along with the appropriate transaction code (i.e., "A").

Trigger Name: TI_STUD_PRG_TRAN_X
Associated Table: STUD_PRG_X
Type Trigger: After Insert

```
Source Code */
declare
 numrows INTEGER;
 rowindex INTEGER;
 trannewrec STUD_PRG_X_TRAN.STUD_PRG_X_TRAN;
 newrec STUD_PRG_X_TRAN.STUD_PRG_X_row;
begin

  if (STUD_PRG_X_TRAN.get_nest_level = 0) then
   STUD_PRG_X_TRAN.increment_nest_level;
   rowindex := 1;
   while rowindex <= STUD_PRG_X_TRAN.get_row_count loop
    STUD_PRG_X_TRAN.TRAN_by_index(rowindex,trannewrec);
    select * into newrec from STUD_PRG_X
    where
     trannewrec.Prg_ID = STUD_PRG_X.Prg_ID and
     trannewrec.StudSSN_ID = STUD_PRG_X.StudSSN_ID and
     trannewrec.PrgEnrollDate = STUD_PRG_X.PrgEnrollDate;

 INSERT INTO STUD_PRG_TRAN_X
   VALUES ('A', newrec.prg_id, newrec.StudSSN_ID, newrec.prgenrolldate);

rowindex := rowindex + 1;
  end loop;
  STUD_PRG_X_TRAN.decrement_nest_level;
 end if;
end;


/*
```

This package specification contains the specification for a series of functions and procedures associated with the two triggers described above. This package allows for the insertion of transaction codes into the STUD_PRG_TRAN_X table upon enrollment of new students.

Package Specification for Package: STUD_PRG_X_TRAN

```
Source Code */

PACKAGE STUD_PRG_X_TRAN IS

/*
*****************************************
*****************************************
```

Package SPECIFICATION STUD_PRG_X_TRAN for creating PL/SQL tables for use
in updating transaction records to avoid mutating tables
By Aaron Slaughter

```
28 April 97
*****************************************
*****************************************
*/
/* used to return rows from PL/SQL tables */
  type STUD_PRG_X_TRAN is record
       (Prg_ID VARCHAR2(6),
        StudSSN_ID VARCHAR2(9),
        PrgEnrollDate DATE);

  /* used to access rows in trigger templates */
  type STUD_PRG_X_row is record
       (Prg_ID VARCHAR2(6),
        StudSSN_ID VARCHAR2(9),
        PrgEnrollDate DATE,
        PrgCCD DATE,
        PrgDipIssueDate DATE,
        SC_Comp CHAR(1),
        Disenrolled CHAR(1));

  /* used in recursive relationship situations */
  function get_nest_level return number;
  procedure increment_nest_level;
  procedure decrement_nest_level;

  /* resets PL/SQL table contents */
  procedure initialize_table;

  /* inserts a row in to PL/SQL table */
  procedure INSERT_TRAN(
    Prg_ID IN VARCHAR2,
    StudSSN_ID IN VARCHAR2,
    PrgEnrollDate IN DATE
  );

  /* determines if there are any more rows in PL/SQL tables */
  function more_rows return boolean;

  /* determines the number of rows in the PL/SQL table */
  function get_row_count return number;

  /* gets the next row and decrements row count thereby deleting row
     in PL/SQL tables */
  procedure next_TRAN (tbl_row out STUD_PRG_X_TRAN);

  /* gets row by index from PL/SQL tables */
  procedure TRAN_by_index(n in number, tbl_row out STUD_PRG_X_TRAN);

end;


/*
```

This package body contains the source code for a series of functions and procedures associated with the two triggers described above. This package allows for the insertion of transaction codes into the STUD_CRS_TRAN_X table upon enrollment of new students.

Package Body for Package: STUD_CRS_X_TRAN

Source Code */

```
PACKAGE BODY    STUD_PRG_X_TRAN IS
/*
*****************************************
*****************************************
Package BODY STUD_PRG_X_TRAN for creating PL/SQL tables for use
in updating transaction records to avoid mutating tables
By Aaron Slaughter
28 April 97
*****************************************
*****************************************
*/
/* declare PL/SQL table types to store VALUES       from STUD_PRG_X for INSERT
  into STUD_PRG_TRANS_X */
 type Prg_ID_typ is table of
   VARCHAR2(6) index by binary_integer;
 type StudSSN_ID_typ is table of
   VARCHAR2(9) index by binary_integer;
 type PrgEnrollDate_typ is table of
   DATE index by binary_integer;

 /* declare PL/SQL tables to store VALUES from STUD_PRG_X for INSERT
    into STUD_PRG_TRANS_X */
 Prg_ID_tbl Prg_ID_typ;
 StudSSN_ID_tbl StudSSN_ID_typ;
 PrgEnrollDate_tbl PrgEnrollDate_typ;


 /* row count in PL/SQL tables */
 nrows number;

 /* nest level */
 TriggerNestLevel number := 0;

 procedure initialize_table is
 begin
   nrows := 0;
   TriggerNestLevel := 0;
 end;

 function get_nest_level return number is
 begin
   return TriggerNestLevel;
 end;

 procedure increment_nest_level is
 begin
   TriggerNestLevel := TriggerNestLevel + 1;
 end;
```

```
 procedure decrement_nest_level is
 begin
  TriggerNestLevel := TriggerNestLevel - 1;
 end;

 procedure INSERT_TRAN(
  Prg_ID IN VARCHAR2,
  StudSSN_ID IN VARCHAR2,
  PrgEnrollDate IN DATE
 ) is
 begin
  nrows := nrows + 1;
  Prg_ID_tbl(nrows) := Prg_ID;
  StudSSN_ID_tbl(nrows) := StudSSN_ID;
  PrgEnrollDate_tbl (nrows) := PrgEnrollDate;


 end;

 function more_rows return boolean is
 begin
  return (nrows > 0);
 end;

 function get_row_count return number is
 begin
  return (nrows);
 end;

 procedure next_TRAN (tbl_row out STUD_PRG_X_TRAN) is
 begin
  tbl_row.Prg_ID := Prg_ID_tbl(nrows);
  tbl_row.StudSSN_ID := StudSSN_ID_tbl(nrows);
  tbl_row.PrgEnrollDate := PrgEnrollDate_Tbl (nrows);

  nrows := nrows - 1;
 end;

 procedure TRAN_by_index (n in number, tbl_row out STUD_PRG_X_TRAN) is
 begin
  if n > 0 and n <= nrows then
    tbl_row.Prg_ID := Prg_ID_tbl(n);
    tbl_row.StudSSN_ID := StudSSN_ID_tbl(n);
    tbl_row.PrgEnrollDate := PrgEnrollDate_tbl(n);

  end if;
 end;

end;
```

III. Student course enrollment upon student program enrollment: The following set of triggers, procedures, and functions allows for the automatic insertion or update of student course enrollments in the STUD_CRS_X table whenever a student is enrolled in the corresponding program in the STUD_PRG_X table.

/*

This trigger allows the PL/SQL table to be initialized (emptied) and the corresponding course values, student SSN, course enrollment date, and course required completion date for all courses associated with the specified program to be read into the PL/SQL table.

Trigger Name: TIROW_STUD_CRS_X
Associated Table: STUD_PRG_X
Type Trigger: For Each Row, After Insert

Source Code: */

```
DECLARE
v_Crs_ID Crs_Prg_X.Crs_ID%Type;
CURSOR Crs_Cursor IS
SELECT Crs_ID
FROM CRS_PRG_X
WHERE Prg_ID = :new.Prg_ID;

BEGIN

PME_PRG_CRS.initialize_table;


 IF (PME_PRG_CRS.get_nest_level = 0) THEN
  FOR Crs_Rec in Crs_Cursor LOOP
   EXIT WHEN Crs_Cursor%NOTFOUND;
                    SELECT Crs_ID
                    INTO v_Crs_ID
            FROM CRS_PRG_X
           WHERE Crs_ID = Crs_Rec.Crs_ID;

            PME_PRG_CRS.INSERT_CRS(
            v_crs_ID,
            :new.StudSSN_ID,
            :new.PrgEnrollDate,
            :new.PrgCCD);

  END LOOP;
 END IF;

END;
```

/*

This trigger allows for all associate STUD_CRS_X instances to be entered or updated in the STUD_CRS_X table for each Program enrollment.

Trigger Name: TIA_NEW_STUD_CRS_X
Associated Table: STUD_PRG_X
Type Trigger: After Insert

Source Code: */

```
DECLARE
rowindex        INTEGER;
crsnewrec       PME_PRG_CRS.PME_PRG_CRS;
newrec          PME_PRG_CRS.STUD_CRS_X_row;
```

```
v_temp              STUD_CRS_X.Crs_ID%type;
BEGIN

IF (PME_PRG_CRS.get_nest_level = 0) then
  PME_PRG_CRS.increment_nest_level;
  rowindex := 1;
  WHILE rowindex <= PME_PRG_CRS.get_row_count LOOP
    PME_PRG_CRS.CRS_by_index (rowindex, crsnewrec);

    INSERT INTO STUD_CRS_X
    VALUES (crsnewrec.Crs_ID, crsnewrec.StudSSN_ID, crsnewrec.CrsEnrollDate,
          crsnewrec.CrsCCD, NULL, NULL, 'Y', 'N', 'N');
    rowindex := rowindex + 1;
  END LOOP;

PME_PRG_CRS.decrement_nest_level;

END IF;

/*
```

This package specification contains the specification for a series of functions and procedures associated with the two triggers described above. This package allows for the insertion of student course enrollment instances into the STUD_CRS_X table upon enrollment of new students into programs.

Package Specification for Package: PME_PRG_CRS

Source Code */

```
PACKAGE    PME_PRG_CRS AS
/*
*****************************************
*****************************************
Package SPECIFICATION  PME_PRG_CRS for creating PL/SQL tables for use
in updating transaction records to avoid mutating tables
By Aaron Slaughter
28 April 97
*****************************************
*****************************************
*/
/* used to return rows from PL/SQL tables */
 type PME_PRG_CRS  is record
      (Crs_ID VARCHAR2(6),
       StudSSN_ID VARCHAR2(9),
       CrsEnrollDate DATE,
       CrsCCD DATE,
       Disenrolled CHAR(1));

 /* used to access rows in trigger templates */
 type STUD_CRS_X_row is record
      (Crs_ID VARCHAR2(6),
       StudSSN_ID VARCHAR2(9),
       CrsEnrollDate DATE,
       CrsCCD DATE,
```

```
        CrsFinalGrade VARCHAR2(3),
        CrCertDate DATE,
        SC_PME CHAR(1),
        SC_Comp CHAR(1),
        Disenrolled CHAR(1));

  /* used in recursive relationship situations */
  function get_nest_level return number;
  procedure increment_nest_level;
  procedure decrement_nest_level;

  /* resets PL/SQL table contents */
  procedure initialize_table;

  /* inserts a row in to PL/SQL table */
  procedure INSERT_CRS(
    v_Crs_ID IN VARCHAR2,
    StudSSN_ID IN VARCHAR2,
    CrsEnrollDate IN DATE,
    CrsCCD IN DATE
  );

  /* determines if there are any more rows in PL/SQL tables */
  function more_rows return boolean;

  /* determines the number of rows in the PL/SQL table */
  function get_row_count return number;

  /* gets the next row and decrements row count thereby deleting row
     in PL/SQL tables */
  procedure next_CRS (tbl_row out PME_PRG_CRS);

  /* gets row by index from PL/SQL tables */
  procedure CRS_by_index(n in number, tbl_row out PME_PRG_CRS);

  /*handle duplicate value exceptions*/
  procedure DUP_VAL (numrows IN INTEGER);

  /*handle no data found exceptions*/
  procedure NO_DATA (numrows IN INTEGER);
end;


/*
```

This package body contains the source code for a series of functions and procedures associated with the two triggers described above. This package allows for the insertion of student course enrollment instances into the STUD_CRS_X table upon enrollment of new students into programs.

Package Body for Package: PME_PRG_CRS

Source Code */

```
PACKAGE BODY    PME_PRG_CRS  IS
```

```
/*
*****************************************
*****************************************
Package BODY PME_PRG_CRS  for creating PL/SQL tables for use
in updating STUD_CRS_X records for PME enrollments  to avoid mutating tables
By Aaron Slaughter
28 April 97
*****************************************
*****************************************
*/
/* declare PL/SQL table types to store VALUES from CRS_PRG_X for INSERT into STUD_CRS_X */
 type Crs_ID_typ is table of
   VARCHAR2(6) index by binary_integer;
 type StudSSN_ID_typ is table of
   VARCHAR2(9) index by binary_integer;
 type CrsEnrollDate_typ is table of
   DATE index by binary_integer;
 type CrsCCD_typ is table of
   DATE index by binary_integer;
 type Disenrolled_typ is table of
  CHAR(1) index by binary_integer;

 /* declare PL/SQL tables to store VALUES from CRS_PRG_X for INSERT into STUD_CRS_X */
 Crs_ID_tbl Crs_ID_typ;
 StudSSN_ID_tbl StudSSN_ID_typ;
 CrsEnrollDate_tbl CrsEnrollDate_typ;
 CrsCCD_tbl CrsCCD_typ;
 Disenrolled_tbl Disenrolled_typ;


 /* row count in PL/SQL tables */
 nrows number;

 /* nest level */
 TriggerNestLevel number := 0;

 procedure initialize_table is
 begin
   nrows := 0;
   TriggerNestLevel := 0;
 end;

 function get_nest_level return number is
 begin
   return TriggerNestLevel;
 end;

 procedure increment_nest_level is
 begin
   TriggerNestLevel := TriggerNestLevel + 1;
 end;

 procedure decrement_nest_level is
 begin
   TriggerNestLevel := TriggerNestLevel - 1;
```

```
end;

procedure INSERT_CRS (
  v_Crs_ID IN VARCHAR2,
  StudSSN_ID IN VARCHAR2,
  CrsEnrollDate IN DATE,
  CrsCCD IN DATE
) is
begin
  nrows := nrows + 1;
  Crs_ID_tbl(nrows) := v_Crs_ID;
  StudSSN_ID_tbl(nrows) := StudSSN_ID;
  CrsEnrollDate_tbl (nrows) := CrsEnrollDate;
  CrsCCD_tbl (nrows) := CrsCCD;
  Disenrolled_tbl (nrows) := 'N';

end;

function more_rows return boolean is
begin
  return (nrows > 0);
end;

function get_row_count return number is
begin
  return (nrows);
end;

procedure NEXT_CRS  (tbl_row out PME_PRG_CRS) is
begin
  tbl_row.Crs_ID := Crs_ID_tbl(nrows);
  tbl_row.StudSSN_ID := StudSSN_ID_tbl(nrows);
  tbl_row.CrsEnrollDate := CrsEnrollDate_tbl (nrows);
  tbl_row.CrsCCD := CrsCCD_tbl (nrows);
  tbl_row.Disenrolled := Disenrolled_tbl (nrows);
  nrows := nrows - 1;
end;

procedure CRS_by_index (n in number, tbl_row out PME_PRG_CRS) is
begin
  if n > 0 and n <= nrows then
    tbl_row.Crs_ID := Crs_ID_tbl(n);
    tbl_row.StudSSN_ID := StudSSN_ID_tbl(n);
    tbl_row.CrsEnrollDate := CrsEnrollDate_tbl(n);
    tbl_row.CrsCCD := CrsCCD_tbl(n);
    tbl_row.Disenrolled := Disenrolled_tbl(n);

  end if;
end;

procedure DUP_VAL (numrows IN INTEGER)is

 rowindex              INTEGER;
 crsnewrec             PME_PRG_CRS;
 newrec                STUD_CRS_X_row;
```

```
 CURSOR Current_Enroll_Cursor IS
  SELECT Crs_ID
  FROM STUD_CRS_X
  WHERE StudSSN_ID = crsnewrec.StudSSN_ID;


BEGIN
  rowindex := numrows;
  IF (get_nest_level = 1) THEN


  WHILE rowindex <= get_row_count LOOP
  CRS_by_index (rowindex, crsnewrec);

      FOR Crs_Rec in Current_Enroll_Cursor LOOP
        EXIT WHEN Current_Enroll_Cursor%NOTFOUND;

              UPDATE STUD_CRS_X
              SET SC_PME = 'Y', CrsEnrollDate = CRSNEWREC.CrsEnrollDate, CrsCCD =
CRSNEWREC.CrsCCD
              WHERE Crs_ID = CRSNEWREC.Crs_ID AND STUDSSN_ID =
CRSNEWREC.STUDSSN_ID;

      END LOOP;


  rowindex := rowindex + 1;
  NO_DATA (rowindex);


  END LOOP;
 END IF;


  EXCEPTION
  WHEN NO_DATA_FOUND
  THEN
  rowindex := rowindex + 1;
  NO_DATA (rowindex);
  WHEN OTHERS
  THEN
  NULL;

END;

procedure NO_DATA (numrows IN INTEGER) is

rowindex          INTEGER;
crsnewrec         PME_PRG_CRS;
newrec            STUD_CRS_X_row;


BEGIN
```

```
  IF (get_nest_level = 1) THEN
  rowindex := numrows;
  WHILE rowindex <= get_row_count LOOP
   CRS_by_index (rowindex, crsnewrec);

     INSERT INTO STUD_CRS_X
     VALUES (crsnewrec.Crs_ID, crsnewrec.StudSSN_ID, crsnewrec.CrsEnrollDate,
          crsnewrec.CrsCCD, NULL, NULL, 'Y', 'N', 'N');

   rowindex := rowindex + 1;

 END LOOP;


 END IF;

 EXCEPTION

 WHEN
  DUP_VAL_ON_INDEX
 THEN
  DUP_VAL (rowindex);
/*
The when others exception is necessary to handle a mutating table anomaly.  Without
the when others exception, a mutating table error is raised for this procedure,
even though the code for this procedure is identical to the code contained in
the TIA_NEW_STUD_CRS trigger.  Subtracting 1 from rowindex and calling DUP_VAL
effectively defeats the mutating table error in this particular case.
Added by Aaron Slaughter on 6 May 1997.*/
 WHEN
  OTHERS
 THEN
   DUP_VAL (rowindex - 1);
 END;
end;
```

IV. Update of Student records upon update or insert of Marine Corps Total Force records: The following trigger allows for the automatic update of students in the STUDENT table whenever an associated MCTF_PERS record is inserted or updated in the MCTF_PERS table

```
/*
```

This trigger allows the update of a student record whenever an associate Marine Corps Total Force record is inserted or updated.

```
Trigger Name: MCTF_PERS_STUD_UPDATE
Associated Table: MCTF_PERS
Type Trigger:  For Each Row,  After Insert or Update

Source Code: */

BEGIN
UPDATE STUDENT
SET RANK = :NEW.MCTFRANK, GRADE = :NEW.MCTFGRADE, SERVCOMP =
:NEW.MCTFCOMP,
STUDLASTNAME = :NEW.MCTFLASTNAME, STUDMI = :NEW.MCTFMIDINIT
```

```
WHERE STUDSSN_ID = :NEW.MCTFSSN;
END;
```

V. Enrollment eligibility check: The following trigger allows for the verification of eligibility requirements upon an attempt to enroll a student into a Program. This trigger is written to demonstrate this functionality for the 8000 series programs. It can be used as a template to write verification triggers for other programs, based on their specific eligibility requirements.

```
/*
```

This trigger allows the verification of student eligibility upon an attempt to enroll a student in the 8000 series program. If the student is eligible, enrollment is allowed; if ineligible, enrollment is disallowed and the appropriate message is displayed to the user.

```
Trigger Name: TIBROW_CHECK_GRADE
Associated Table: STUD_PRG_X
Type Trigger:  For Each Row,  Before Insert

Source Code: */

DECLARE
v_TEMP          STUD_CRS_X.Crs_ID%type;
V_GRADE    GRADE.GRADE%TYPE;
V_SELGRADE  GRADE.GRADE%TYPE;
V_SSN     STUDENT.STUDSSN_ID%TYPE;
V_COMP            STUDENT.SERVCOMP%TYPE;
V_PROG            PROGRAM.PRG_ID%TYPE;
V_GRADEUSMC PROGRAM.P_GRADEELUSMC%TYPE;
V_GRADEOTH  PROGRAM.P_GRADEELOTHER%TYPE;

CURSOR  NEW_ENROLL_CURSOR IS
 SELECT STUDSSN_ID
 FROM STUDENT
 WHERE STUDSSN_ID = :new.STUDSSN_ID;

CURSOR CHECK_PROG_CURSOR IS
 SELECT PRG_ID
 FROM PROGRAM
 WHERE PRG_ID = :NEW.PRG_ID;


BEGIN

 FOR STUD_REC IN NEW_ENROLL_CURSOR LOOP
  EXIT WHEN NEW_ENROLL_CURSOR%NOTFOUND;
  SELECT GRADE, SERVCOMP, STUDSSN_ID
  INTO V_GRADE, V_COMP, V_SSN
  FROM STUDENT
  WHERE STUDSSN_ID = STUD_REC.STUDSSN_ID;

  FOR PROG_REC IN CHECK_PROG_CURSOR LOOP
   EXIT WHEN CHECK_PROG_CURSOR%NOTFOUND;
   SELECT PRG_ID, P_GradeElUSMC, P_GradeElOther
   INTO V_PROG, V_GRADEUSMC,V_GRADEOTH
   FROM PROGRAM
   WHERE PRG_ID = PROG_REC.PRG_ID;
```

```
    IF  V_COMP  IN ('1', '2', '3', '4') THEN
     IF V_GRADE < V_GRADEUSMC THEN
      STUD_PRG_X_new.initialize_table;
      STUD_PRG_X_old.initialize_table;
      PME_PRG_CRS.initialize_table;
      raise_application_error(
      -20002,
      'Cannot INSERT "STUD_PRG_X" because "STUDENT"
is not of high enough grade. Marines desiring to enroll in this program must be of
grade '||V_GRADEUSMC ||'  or higher.




,

       );
     END IF;

    ELSIF  V_COMP NOT IN ('1', '2', '3', '4') THEN
     IF V_GRADE < V_GRADEOTH THEN
      STUD_PRG_X_new.initialize_table;
      STUD_PRG_X_old.initialize_table;
      PME_PRG_CRS.initialize_table;
      raise_application_error(
      -20002,
      'Cannot INSERT "STUD_PRG_X" because "STUDENT"
is not of high enough grade. Non-Marines desiring to enroll in this program must
be of grade '||V_GRADEOTH ||'  or higher.




,

       );
     END IF;

   END IF;
  END LOOP;
 END LOOP;

END;
```

APPENDIX S. ATTRIBUTE AVAILABILITY REPORT

The following legend explains the codes used by Mr. Joseph Rudd when completing the attribute availability report. The attribute availability report is sorted by Entity and attribute within that entity.

LEGEND:

AV - Short for available. (Y = yes, N= no, M= Maybe) Most of the "M's" are for data that is available but not in digital format. I t will have to be entered by hand.

Column Name - This column contains the entity names (bold) and the attributes for that entity.

Key - Each key attribute is labeled with a code:

P = the primary key for the attribute
P(d) = the primary key for the domain table
F = a foreign key
F(d) = a foreign key that references a domain table.
P/F = Primary and Foreign key. Typically part of a multi-part key.

Column Datatype - Datatype from ER/Win.

Curr Type - This column is only filled for attributes with a difference between the current data and the model. An entry of 'X(8)' means that the current data type is alphanumeric and the length is 8 bytes.

Null Option - From ER/Win

Source - Codes are used here to save space. The normal code will look like : "M/mcourse/title" . The first field is for the database, the second is for the dataset within that database and the third is the item. If the data is available from more than one I used " & " to separate the locations.
The databases are :

- M = mcidb.dbase.mciais
- A = ansref.dbase.mciais
- An "F" as the first value represents a Flat file as the source.
- "ACL" means that the information can be found in the MCI Annual Course listing.

Comments - Entered by Mr. Rudd for clarification.

AV	Column Name	Key	Column Datatype	Curr Type	Null Option	Source	Comments
	ANSWER_CODE						
N	AnsCode_ID	P(d)	VARCHAR2(1)		NOT NULL		Though not available from our database, the values should be easy to get from our programs
N	AnsCodeDesc		VARCHAR2(30)		NOT NULL		
	CATEGORY_CODE						
N	CatCode_ID	P(d)	VARCHAR2(2)		NOT NULL		Though not available from our database, the values should be easy to get from our programs
N	CatCodeDesc		VARCHAR2(20)		NOT NULL		
	COMPONENT_LINE_ITEM						
N	CompLineItm_ID	P	NUMBER		NOT NULL		
N	InvNo_ID	P/F	NUMBER		NOT NULL		
N	Comp_ID	P/F	VARCHAR2(10)		NOT NULL		
N	CompQty		NUMBER		NOT NULL		
	COMPONENTS						
N	Comp_ID	P	VARCHAR2(10)		NOT NULL		This data may be avail from logs.
M	CompName		VARCHAR2(30)		NOT NULL		This data may be avail from logs.
M	CompDesc		VARCHAR2(60)		NOT NULL		This data may be avail from logs.
M	CompAvail		CHAR(1)		NULL		This data may be avail from logs.
	COURSE						
Y	Crs_ID	P	VARCHAR2(6)		NOT NULL	M/mcourse/course	
N	RemMeth_ID	F(d)	VARCHAR2(10)		NOT NULL		We do not have a 'table' with the required values, however we should be able to extract the different types

AV	Column Name	Key	Column Datatype	Curr Type	Null Option	Source	Comments
							of remediation from the programs. Do we really want a 10 byte ID ?
Y	CrsNo		VARCHAR2(4)		NOT NULL	M/mcourse/vcourse	
Y	StatCode_ID	F(d)	VARCHAR2(2)		NOT NULL	M/mcourse/status	
Y	CrsTitle		VARCHAR2(60)	X(80)	NOT NULL	M/mcourse/title	
Y	CrsAbbr		VARCHAR2(20)		NOT NULL	M/mcourse/crs-abbr	
Y	CrsDateOpen		DATE	X(6)	NULL	M/mcourse/date-crs-open	
M	CrsStudyHours		VARCHAR2(3)	???	NULL	ACL	** Some of the courses have half hour parts. E.G. course 7205 has 9.5 study hours.
Y	CrsDateClsd		DATE	X(6)	NULL	M/mcourse/date-crs-close	
Y	CrsResCred		CHAR(8)	X(2)	NOT NULL	M/mcourse/reserve-credits	Currently it is 2 bytes, I do not think we need 8. I think that 3 bytes will be enough. Can be null.
M	EdLev	F(d)	VARCHAR2(4)		NULL	ACL	
M	CrsCreditHrs		VARCHAR2(3)		NULL	ACL	
M	CrsDesc		LONG		NULL	ACL	
M	CrsDesignedFor		VARCHAR2(200)		NULL	ACL	
N	CrsOnHandQty		NUMBER		NOT NULL		
N	C_GradePre		CHAR(2)		NULL	ACL & current programs.	
Y	C_PME		CHAR(1)	x(2)	NOT NULL	Where value of course = "xx00".	Can be 1 byte, however for data conversion we must be aware that it is currently 2 byte.
Y	C_Accr		CHAR(1)	x(2)	NOT NULL	M/mcourse/accredited	Can be 1 byte, however for data conversion we must be aware that it is currently 2 byte.
	COURSE_COMPONENTS_X						
Y	Crs_ID	P/F	VARCHAR2(6)		NOT		

AV	Column Name	Key	Column Datatype	Curr Type	Null Option	Source	Comments
					NULL		
M	Comp_ID	P/F	VARCHAR2(10)		NOT NULL		
N	CC_Qty		NUMBER		NOT NULL		
	COURSE_LINE_ITEM						
N	CrsLineItm_ID	P	NUMBER		NOT NULL		
N	InvNo_ID	P/F	NUMBER		NOT NULL		
Y	Crs_ID	P/F	VARCHAR2(6)		NOT NULL		
N	CrsQty		NUMBER		NOT NULL		
	COURSE_PREREQ						
N	JrCrs_ID	P/F	VARCHAR2(6)		NOT NULL		
N	SrCrs_ID	P/F	VARCHAR2(6)		NOT NULL		
	COURSE_PROGRAM_X						
N	Crs_ID	P/F	VARCHAR2(6)		NOT NULL		Not avail from data, however it can be figured from the Crs_Id values.
N	Prg_ID	P/F	VARCHAR2(6)		NOT NULL		
	CUSTOMER						
N	CustSSN_ID	P	CHAR(9)		NOT NULL		
N	State	F(d)	CHAR(2)		NOT NULL		
N	CustLastName		VARCHAR2(20)		NOT NULL		
N	CustFirstName		VARCHAR(10)		NOT NULL		
N	CustMidInit		CHAR(1)		NULL		
N	CustDSN		CHARACTER(7)		NULL		Need to be able to accept an extension also. Either enlarge this attrib. or create another attrib.

AV	Column Name	Key	Column Datatype	Curr Type	Null Option	Source	Comments
N	CustCommNo		CHAR(10)		NULL		Need to be able to accept an extension also.
N	CustAddr1		VARCHAR2(46)		NOT NULL		
N	CustAddr2		VARCHAR2(46)		NOT NULL		
N	CustCity		VARCHAR2(34)		NOT NULL		
N	CustZip		VARCHAR2(9)		NOT NULL		
	ED_LEVEL						
M	EdLev	P(d)	VARCHAR2(4)		NOT NULL	ACL	
N	EdLevDesc		VARCHAR2(20)		NOT NULL		
	ERROR_CODES						
N	ErrCode_ID	P(d)	VARCHAR2(1)		NOT NULL		Though not available from our database, the values should be easy to get from our programs
N	ErrCodeDesc		VARCHAR2(55)		NOT NULL		Though not available from our database, the values should be easy to get from our programs
	ERROR_LISTING						
N	ErrCntr_ID	P	NUMBER		NOT NULL		
Y	ErrDate	P	DATE		NOT NULL		Avail from scanned data
Y	Crs_ID	F	VARCHAR2(6)		NOT NULL	F/scanned data	If scanned properly
M	Exam_ID	F	NUMBER		NOT NULL	F/scanned data & student_course-exam_x	
N	ErrCode_ID	F	VARCHAR2(1)		NOT NULL		
Y	StudSSN_ID	F	VARCHAR2(9)		NULL	F/scanned data	If scanned properly & student
Y	ScanSSN_ID		VARCHAR2(9)		NOT NULL	F/scanned data	If scanned properly

AV	Column Name	Key	Column Datatype	Curr Type	Null Option	Source	Comments
Y	OpscanNumber		VARCHAR2(4)		NOT NULL	F/scanned data	
	EXAM						
M	Exam_ID	P	NUMBER	x(2)	NOT NULL		** the exam-id is currently a letter. I think we will want to keep it as a letter vice a system generated number. We will need to be able to reference the exam by its current letter to populate the reference attributes.
Y	Crs_ID	P/F	VARCHAR2(6)		NOT NULL		
Y	CatCode_ID	F(d)	VARCHAR2(2)		NOT NULL		Only avail from "mcourse" file, not by exam.
Y	ExmDateOpen		DATE	x(6)	NULL	M/mcourse/date-ex1-open & date-ex2-open	
N	ExmDateClose		DATE	x(6)	NULL		
N	ExmPassScore		NUMBER		NOT NULL		Not avail as a data item in our current system, however the values should be easy to fill from the programs.
Y	ExNoQuest		NUMBER		NOT NULL	A/ansref-detl/num-answers	
	EXAM_COMPONENT_X						
N	Comp_ID	P/F	VARCHAR2(10)		NOT NULL		
M	Exam_ID	P/F	NUMBER		NOT NULL		
Y	Crs_ID	P/F	VARCHAR2(6)		NOT NULL		
N	ExmCompQty		NUMBER		NOT NULL		
	GRADE						
N	Grade	P(d)	CHAR(2)		NOT NULL		
N	GradeDesc		VARCHAR2(20)		NOT NULL		

AV	Column Name	Key	Column Datatype	Curr Type	Null Option	Source	Comments
	INVOICE						
N	InvNo_ID	P	NUMBER		NOT NULL		
N	ClerkSSN_ID	P/F	CHAR(9)		NOT NULL		
N	CustSSN_ID	P/F	CHAR(9)		NOT NULL		
N	CntrNo	P/F	NUMBER		NOT NULL		
	ISSUE						
N	Issue_ID	P	NUMBER		NOT NULL		
N	IssueAbbr		VARCHAR2(30)		NOT NULL		
N	IssueDesc		VARCHAR2(2000)		NOT NULL		
N	IssueActionRec		VARCHAR2(2000)		NULL		
	ISSUE_CALL						
N	CntrNo	P	NUMBER		NOT NULL		
N	CustSSN_ID	P/F	CHAR(9)		NOT NULL		
N	ClerkSSN_ID	P/F	CHAR(9)		NOT NULL		
N	Issue_ID	F	NUMBER		NOT NULL		
N	InstToCaller		LONG		NULL		
N	FollowUpTime		DATE		NULL		
N	ActionReq		VARCHAR2(2000)		NOT NULL		
N	IssueCallComments		VARCHAR2(2000)		NOT NULL		
	ISSUE_CALL_STUDENT_X						
N	CntrNo	P	NUMBER		NOT NULL		
N	StudSSN_ID	P/F	VARCHAR2(9)		NOT		

AV	Column Name	Key	Column Datatype	Curr Type	Null Option	Source	Comments
					NULL		
N	CustSSN_ID	P/F	CHAR(9)		NOT NULL		
N	ClerkSSN_ID	P/F	CHAR(9)		NOT NULL		
	JOB_AID						
N	JobAid_ID	P	VARCHAR2(10)		NOT NULL		
N	JAName		VARCHAR2(30)		NOT NULL		
N	JADesc		VARCHAR2(600)		NOT NULL		
N	JAOnHandQty		NUMBER		NOT NULL		
	JOB_AID_COMPONENTS_X						
N	Comp_ID	P/F	VARCHAR2(10)		NOT NULL		
N	JobAid_ID	P/F	VARCHAR2(10)		NOT NULL		
N	JACompQty		NUMBER		NOT NULL		
	JOB_AID_LINE_ITEM						
N	JALineItem_ID	P	NUMBER		NOT NULL		
N	InvNo_ID	P/F	NUMBER		NOT NULL		
N	JobAid_ID	P/F	VARCHAR2(10)		NOT NULL		
N	JAQty		NUMBER		NOT NULL		
	MCTF_PERS						
Y	MCTFSSN	P	CHAR(9)		NOT NULL	F/vefaddr	
Y	StudSSN_ID	F	VARCHAR2(9)		NULL	F/vefaddr	
Y	MCTFLastName		VARCHAR2(20)		NOT NULL	F/vefaddr	
Y	MCTFFirstName		VARCHAR2(1		NOT	F/vefaddr	

AV	Column Name	Key	Column Datatype	Curr Type	Null Option	Source	Comments
			0)		NULL		
Y	MCTFMidInit		VARCHAR2(1)	X(2)	NULL	F/vefaddr	This is actually the 1st & Middle initial. That is the way that is stored on the mainframe. We can of course just grab the middle initial during the data load.
Y	MCTFMOS		CHAR(4)		NOT NULL	F/vefaddr	Currently, I pull both Primary and Billet MOS.
Y	MCTFPlatCode		CHAR(4)		NOT NULL	F/vefaddr	
Y	MCTF_RUCMCC	F	CHAR(8)		NOT NULL	F/vefaddr	This will have to be VARCHAR(8) because reservists do not always have an MCC.
Y	MCTFComp		CHAR(2)		NOT NULL	F/vefaddr	MCI local use value.
M	MCTFRank		VARCHAR2(6)		NOT NULL		Not currently pulled from MCTFS, but it can be.
Y	MCTFGrade		CHAR(2)		NOT NULL	F/vefaddr	
M	MCTF_ECC		DATE		NULL		Currently I pull EAS, a 9 byte text value.
M	MCTFSelToGrade		CHAR(2)		NULL		Not currently pulled from MCTFS, but it can be.
Y	MCTFAddr1		VARCHAR2(46)		NOT NULL	F/vefaddr	Can be null. I only pull this for Reserve Marines.
Y	MCTFAddr2		VARCHAR2(46)		NULL	F/vefaddr	
Y	MCTFCity		VARCHAR2(34)		NULL	F/vefaddr	
Y	MCTFZip		VARCHAR2(9)		NOT NULL	F/vefaddr	Can be null. I only pull this for Reserve Marines.
Y	MCTFState		VARCHAR2(2)		NULL	F/vefaddr	
	MULTIPLE_CHOICE_EXAM						
N	QuestNo	P	NUMBER		NOT NULL		
N	Exam_ID	P/F	NUMBER		NOT		

AV	Column Name	Key	Column Datatype	Curr Type	Null Option	Source	Comments
					NULL		
Y	Crs_ID	P/F	VARCHAR2(6)		NOT NULL		
Y	AnsCode_ID	F	VARCHAR2(1)		NOT NULL	A/ansref-detl/answers	This is going to be fun. The data is currently in a 160 byte array. One array for each exam.
Y	ExamRef_1		VARCHAR2(10)		NOT NULL	A/ansref-detl/references	This is going to be fun..... call me.
Y	ExamRef_2		VARCHAR2(10)		NULL	A/ansref-detl/references	This is going to be fun..... call me.
	NON_MARINE_STUDENT_ADDRESS						
Y	StudSSN_ID	P	VARCHAR2(9)		NOT NULL	M/stud-info-detl/SSN	
Y	State	F	CHAR(2)	x(20)	NULL	M/stud-info-detl/state	
Y	NMSAddr1		VARCHAR2(46)	x(40)	NOT NULL	M/stud-info-detl/address1	
N	NMSAddr2		VARCHAR2(46)		NULL		Attrib does not currently exist
Y	NMSCity		VARCHAR2(36)		NULL	M/stud-info-detl/city-st	
Y	NMSZip		VARCHAR2(9)		NOT NULL	M/stud-info-detl/zipcode	
	PROG_COMPONENT_X						
Y	Prg_ID	P/F	VARCHAR2(6)		NOT NULL		
N	Comp_ID	P/F	VARCHAR2(10)		NOT NULL		
N	PrgCompQty		NUMBER		NOT NULL		
	PROG_PREREQ						
N	SrProg_ID	P/F	VARCHAR2(6)		NOT NULL		Not avail from database, however, it can be found within the current programs.
N	JrProg_ID	P/F	VARCHAR2(6)		NOT NULL		Not avail from database, however, it can be found within the current programs.

AV	Column Name	Key	Column Datatype	Curr Type	Null Option	Source	Comments
	PROGRAM						
Y	Prg_ID	P	VARCHAR2(6)		NOT NULL	M/mcourse/"value of xx00"	
Y	PrgNo		VARCHAR2(4)		NOT NULL	M/mcourse/"value of xx00"	PME programs do not have versions.
N	RemMeth_ID	F(d)	VARCHAR2(10)		NOT NULL		
Y	StatCode_ID	F(d)	VARCHAR2(2)		NOT NULL	M/mcourse/status	
Y	PrgTitle		VARCHAR2(60)	x(80)	NOT NULL	M/mcourse/title	
Y	ProgAbbr		VARCHAR2(20)		NOT NULL	M/mcouse/crs-abbr	
Y	PrgDateOpened		DATE	x(6)	NULL	M/mcourse/date-crs-open	
Y	PrgDateClosed		DATE	x(6)	NULL	M/mcourse/date-crs-close	
N	PrgStudyHrs		VARCHAR2(3)		NULL	ACL	
M	EdLev	F(d)	VARCHAR2(4)		NULL	ACL	
N	PrgCreditHrs		VARCHAR2(3)		NULL	ACL	
N	SchoolCode		CHAR(3)		NOT NULL		Values are hard coded in the current programs.
N	PrgOnHandQty		NUMBER		NOT NULL		
N	P_GradePre		CHAR(2)		NULL	ACL	
Y	P_Accr		CHAR(1)	x(2)	NULL	M/mcourse/accredited	Currently 2 bytes, but 1 will do. Need to keep this in mind though during migration.
	PROGRAM_LINE_ITEM						
N	PrgLineItem_ID	P	NUMBER		NOT NULL		
N	InvNo_ID	P/F	NUMBER		NOT NULL		
Y	Prg_ID	P/F	VARCHAR2(6)		NOT NULL		
N	PrgQty		NUMBER		NOT NULL		

AV	Column Name	Key	Column Datatype	Curr Type	Null Option	Source	Comments
	RANK						
N	Rank	P(d)	VARCHAR2(6)		NOT NULL		
N	RankDesc		VARCHAR2(45)		NOT NULL		
	REMEDIATION_METHOD						
N	RemMeth_ID	P(d)	VARCHAR2(10)		NOT NULL		Does this need to be 10 bytes
N	RemMethDesc		VARCHAR2(30)		NOT NULL		
	REPORTING_UNIT_CODE						
Y	MCTF_RUCMCC	P	CHAR(8)	VARCHAR(8)	NOT NULL	M/mruc/rucmcc	Reserve units only have a RUC. The RUC is 5 bytes. We do need to maintain this because of UARS.
Y	UnitName		VARCHAR2(45)		NOT NULL	M/mruc/addr1	data can also come from flat file download, this is where it will come from for future updates.
N	RUCAddr1		VARCHAR2(46)		NOT NULL		
Y	RUCAddr2		VARCHAR2(46)		NULL	M/mruc/addr2	
Y	RUCCityState		VARCHAR2(36)		NOT NULL	M/mruc/city	source data contains both city and state
Y	RUCZip		VARCHAR2(9)		NOT NULL	M/mruc/zipcode	
	SERVICE_COMPONENT						
N	ServComp	P(d)	CHAR(2)		NOT NULL		
N	ServCompDesc		VARCHAR2(40)		NOT NULL		
	SSD_CLERK						
N	ClerkSSN_ID	P	CHAR(9)		NOT NULL		
N	ClerkLastName		VARCHAR2(20)		NOT NULL		
N	ClerkFirstName		CHAR(1)		NOT NULL		

AV	Column Name	Key	Column Datatype	Curr Type	Null Option	Source	Comments
N	ClerkMidInit		CHAR(1)		NOT NULL		
	STATE						
N	State	P(d)	CHAR(2)		NOT NULL		
N	StateDesc		VARCHAR2(20)		NOT NULL		
	STATUS_CODE						
N	StatCode_ID	P(d)	VARCHAR2(2)		NOT NULL		
N	StatCodeDesc		VARCHAR2(60)		NOT NULL		
	STUD_CRS_TRAN_X						
Y	Crs_ID	P/F	VARCHAR2(6)		NOT NULL		
Y	StudSSN_ID	P/F	VARCHAR2(9)		NOT NULL		
Y	Tran_ID	P/F(d	CHAR(1)	x(2)	NOT NULL	M/stud-crs-detl-a & stud-crs-detl-h/lasttran & tranarray	Currently, the lasttran code and the date are stored together in an 8 byte field. The data in the tranarry is a 7 byte repeating field. The data in lasttran is 8 bytes.
Y	SC_TransDate_ID	P	DATE	x(6)	NOT NULL	M/stud-crs-detl-a & stud-crs-detl-h/lasttran & tranarray	Currently, the lasttran code and the date are stored together in an 8 byte field. The data in the tranarry is a 7 byte repeating field. The data in lasttran is 8 bytes.
	STUD_PRG_TRAN_X						
Y	Tran_ID	P/F(d	CHAR(1)		NOT NULL		Currently, the lasttran code and the date are stored together in an 8 byte field. The data in the tranarry is a 7 byte repeating field. The data in lasttran is 8 bytes.
Y	Prg_ID	P/F	VARCHAR2(6)		NOT NULL		
Y	StudSSN_ID	P/F	VARCHAR2(9)		NOT NULL		

AV	Column Name	Key	Column Datatype	Curr Type	Null Option	Source	Comments
Y	SP_TransDate_ID	P	DATE		NOT NULL		Currently, the lasttran code and the date are stored together in an 8 byte field. The data in the tranarry is a 7 byte repeating field. The data in lasttran is 8 bytes.
	STUDENT						
Y	StudSSN_ID	P	VARCHAR2(9)		NOT NULL	M/stud-info-detl/ssn	
Y	Rank	F(d)	VARCHAR2(6)		NULL	M/stud-info-detl/rank	
Y	Grade	F(d)	CHAR(2)		NULL	M/stud-info-detl/grade	
Y	ServComp	F(d)	CHAR(2)		NOT NULL	M/stud-info-detl/component	
Y	StudLastName		VARCHAR2(20)		NOT NULL	M/stud-info-detl/lastname	
N	StudFirstName		VARCHAR2(10)		NOT NULL		We currently only have the students initials in the database.
Y	StudMI		VARCHAR2(1)		NULL	M/stud-info-detl/initials	Both first and middle initials are stored in Initials on MCIDB. Must separate during migration
	STUDENT_ANSWERS						The data for this is avail. only if the record was properly scanned.
Y	StudSSN_ID	P/F	VARCHAR2(9)		NOT NULL	Scanned Data	
M	Exam_ID	P/F	NUMBER		NOT NULL	Scanned Data	
Y	Crs_ID	P/F	VARCHAR2(6)		NOT NULL	Scanned Data	
Y	ExmAnsStrng		VARCHAR2(160)		NOT NULL	Scanned Data	
	STUDENT_COURSE_EXAM_X						
Y	StudSSN_ID	P/F	VARCHAR2(9)		NOT NULL		
M	Exam_ID	P/F	NUMBER		NOT NULL		
Y	Crs_ID	P/F	VARCHAR2(6)		NOT		

AV	Column Name	Key	Column Datatype	Curr Type	Null Option	Source	Comments
					NULL		
Y	ExmDateRecv		DATE	x(6)	NULL	M/stud-crs-detl-h/"date of lasttran or "z" tran.	The "Z" transaction is or a completion, the date is the data received. We can use system date for incoming records in the future.
Y	ExmScore		NUMBER		NULL	M/stud-crs-detl-h/ex1score & ex2score.	In the future will be updated from the grading process.
	STUDENT_COURSE_X						
Y	Crs_ID	P/F	VARCHAR2(6)		NOT NULL		
Y	StudSSN_ID	P/F	VARCHAR2(9)		NOT NULL		
Y	CrsEnrollDate		DATE	x(6)	NOT NULL	M/stud-crs-detl-a & stud-crs-detl-h/enroll-date	
Y	CrsCCD		DATE	x(6)	NULL	M/stud-crs-detl-a & stud-crs-detl-h/deadline	
Y	CrsFinalGrade		VARCHAR2(3)		NULL	M/stud-crs-detl-a & stud-crs-detl-h/ ex1score & ex2score.	
Y	CrCertDate		DATE	x(6)	NULL		The completion certificate should be issued the day of completion. Do we need to track this ? Where did this attrib come from?
Y	SC_PME		CHAR(1)	x(2)	NULL	M/stud-crs-detl-a & stud-crs-detl-h/pme-flag	Currently stored as 2 bytes, can be one.
Y	SC_Comp		CHAR(1)		NULL	M/stud-crs-detl-h/"value of "z" in the lasttran or tranarray.	
	STUDENT_PROGRAM_X						
Y	Prg_ID	P/F	VARCHAR2(6)		NOT NULL		
Y	StudSSN_ID	P/F	VARCHAR2(9)		NOT NULL		

AV	Column Name	Key	Column Datatype	Curr Type	Null Option	Source	Comments
Y	PrgEnrollDate		DATE	x(6)	NOT NULL	M/stud-crs-detl-a & stud-crs-detl-h/enroll-date	
Y	PrgCCD		DATE	x(6)	NULL	M/stud-crs-detl-a & stud-crs-detl-h/deadline	
Y	PrgDipIssueDate		DATE	x(6)	NULL		The diploma should be issued the day of completion. Do we need to track this ? Where did this attrib come from?
Y	SP_Comp		CHAR(1)		NULL	M/stud-crs-detl-h/"value of "z" in the lasttran or tranarray.	
	TRANSACTION_CODES						
N	Tran_ID	P(d)	CHAR(1)		NOT NULL		
N	TransDesc		VARCHAR2(40)		NOT NULL		Recommend increasing the size to 200 (or more) to allow for more detailed descriptions of the transactions.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5000

3. Director, Training and Education 1
   MCCDC, Code C46
   1019 Elliot Rd.
   Quantico, Virginia 22134-5027

4. Director, Marine Corps Research Center 2
   MCCDC, Code C40RC
   2040 Broadway Street
   Quantico, Virginia 22134-5107

5. Director, Studies and Analysis Division 1
   MCCDC, Code C45
   300 Russell Road
   Quantico, Virginia 22134-5130

6. Marine Corps Representative 1
   Naval Postgraduate School
   Code 037, Bld. 234, HA-220
   699 Dyer Road
   Monterey, CA 93943-5000

7. Marine Corps Tactical Systems Support Activity 1
   Technical Advisory Branch
   Attn: Major J.C. Cummiskey
   Box 5555171
   Camp Pendleton, CA 92055-5080

8. Professor Magdi N. Kamel 2
   Department of Systems Management
   Naval Postgraduate School
   Monterey, CA 93943-5000

9. Professor Suresh Sridhar 1
   Department of Systems Management
   Naval Postgraduate School
   Monterey, CA 993943-5000

10. The Marine Corps Institute 2
    Attn: Maj. Lloyd J. Hamashin, USMC
    Washington Navy Yard, 912 Poor St. S.E.,
    Washington, D.C. 20391-5680

11. Maj. Aaron T. Slaughter, USMC 2
    2205 Danforth Court
    Columbia, MO 65201

*U.S. GOVERNMENT PRINTING OFFICE:1998-783-56

MARINE CORPS INSTITUTE
STUDENT SERVICES DIVISION (SSD)
IDEF1X DIAGRAM For HIGH LEVEL VIEW

VERSION 97050 (19 February 1997)
FILE NAME HiLevV97050.ER1
mci2/c:/mci/data model

DEVELOPED BY MAJ. A.T. SLAUGHTER

APPENDIX H. MCI ENTERPRISE LEVEL DATA MODEL

MCTF_PERS
MCTFSSN: CHAR(9) NOT NULL
LastName: VARCHAR2(20) NOT NULL
FirstName: VARCHAR2(10) NOT NULL
MidInit: VARCHAR2(1) NULL
MOS: CHAR(4) NOT NULL
PlatoonCode: CHAR(4) NOT NULL
Comp: CHAR(2) NOT NULL
Rank: VARCHAR2(6) NOT NULL
Grade: CHAR(2) NOT NULL
ECC: DATE NULL
SelToGrade: CHAR(2) NULL
Addr1: VARCHAR2(46) NOT NULL
Addr2: CHAR(46) NULL
City: VARCHAR2(34) NULL
ZipCode: VARCHAR2(9) NOT NULL
State: VARCHAR2(2) NULL

TRAINING
TrainingProg_ID: NUMBER NOT NULL
Description: VARCHAR2(2000) NOT NULL

relates_to

may_be_a

EVENTS
Event_ID: NUMBER NOT NULL
EventName: VARCHAR2(100) NOT NULL
EventDate: DATE NOT NULL
EventTime: VARCHAR2(6) NOT NULL
EventLocation: VARCHAR2(50) NOT NULL
EventCoordinator: VARCHAR2(20) NOT NULL
EventDesc: VARCHAR2(2000) NOT NULL

has_many

MCI_PERSONNEL
SSN_ID: VARCHAR2(9) NOT NULL
MCTFSSN: CHAR(9) NULL (FK)
LastName: VARCHAR2(20) NOT NULL
FirstName: VARCHAR2(10) NOT NULL
Rank: VARCHAR2(6) NOT NULL
DCTB: DATE NOT NULL
ECC: DATE NULL
PDD: DATE NOT NULL
Department: VARCHAR2(20) NOT NULL
MOS: VARCHAR2(4) NOT NULL
JobDescription: VARCHAR2(2000) NULL

1

WAREHOUSE
Werehouse_ID: NUMBER NOT NULL
BldgNumber: VARCHAR2(3) NOT NULL
Department: VARCHAR2(6) NOT NULL
Menager: VARCHAR2(40) NOT NULL
Phone: VARCHAR2(7) NOT NULL

contain

INVENTORY
Warehouse_ID: NUMBER NOT NULL (FK)
Comp_ID: VARCHAR2(10) NOT NULL (FK)
Quantity: NUMBER NOT NULL

may_be_e

PURCHASE
Order_ID: NUMBER NOT NULL
OrderDate: DATE NOT NULL
OrdStatDate: DATE NOT NULL
Supplier: VARCHAR2(100) NOT NULL
EDD: DATE NOT NULL
RDD: DATE NULL
SalesRep: VARCHAR2(40) NULL
Comments: VARCHAR2(2000) NULL

are_tracked_by

is_for

PROGRAM_DEVELOPERS
SSN_ID: VARCHAR2(9) NOT NULL (FK)
Section: VARCHAR2(20) NOT NULL

develops

COURSE_DEVELOPERS
SSN_ID: VARCHAR2(9) NOT NULL (FK)
Section: VARCHAR2(20) NOT NULL
OccFieldSpec: VARCHAR2(4) NOT NULL

develops

is_comprised_of

develops

COMPONENTS
Comp_ID: VARCHAR2(10) NOT NULL
CompName: VARCHAR2(30) NOT NULL
CompDesc: VARCHAR2(60) NOT NULL

JOB_AID
JobAid_ID: VARCHAR2(10) NOT NULL
JobAidName: VARCHAR2(30) NOT NULL
JobAidDesc: VARCHAR2(600) NOT NULL

is_part_of

is_part_of

is_part_of

may_be_composed_of

SSD_PERSONNEL
SSN_ID: VARCHAR2(9) NOT NULL (FK)
Section: VARCHAR2(20) NOT NULL

resolves_an

ISSUE_COMPLAINT
IssueNumber: NUMBER NOT NULL
CustSSN_ID: CHAR(9) NOT NULL (FK)
Date: DATE NOT NULL
Description: VARCHAR2(400) NOT NULL
Comments: VARCHAR2(2000) NOT NULL

generates_a

CUST
CustSSN_ID: CHAR(9) NOT NULL
LastName: VARCHAR2(20) NOT NULL
FirstName: VARCHAR2(10) NOT NULL
MidInit: CHAR(1) NULL
DSN: CHARACTER(7) NULL
CommNo: CHAR(10) NULL
Addr1: VARCHAR2(46) NOT NULL
Addr2: VARCHAR2(46) NULL
City: VARCHAR2(34) NOT NULL
State: CHAR(2) NOT NULL
ZipCode: VARCHAR2(9) NOT NULL

places_an

ORDER
InvoiceNumber: NUMBER NOT NULL
CustSSN_ID: CHAR(9) NOT NULL (FK)
Date: DATE NOT NULL

is_comprised_of

COPY_MATERIAL
Material_ID: NUMBER NOT NULL
Comp_ID: VARCHAR2(10) NOT NULL (FK)
CopyMaterial: LONG NULL
Pointer: VARCHAR2(20) NULL

is_part_of

EXAM
Exam_ID: NUMBER NOT NULL
Crs_ID: VARCHAR2(6) NOT NULL (FK)
DateOpened: DATE NULL
DateClosed: DATE NULL
PassScore: NUMBER NOT NULL
NumberOfQuest: NUMBER NOT NULL

PROGRAM
Program_ID: VARCHAR2(6) NOT NULL
Title: VARCHAR2(60) NOT NULL
ProgAbbr: VARCHAR2(20) NOT NULL
DateOpened: DATE NULL
DateClosed: DATE NULL
StudyHours: VARCHAR2(3) NULL
CreditHours: VARCHAR2(3) NULL
SchoolCode: CHAR(3) NOT NULL
OnHand: NUMBER NOT NULL

is_part_of

COURSE
Crs_ID: VARCHAR2(6) NOT NULL
CourseNumber: VARCHAR2(4) NOT NULL (IE1)
Title: VARCHAR2(60) NOT NULL
CrsAbbr: VARCHAR2(20) NOT NULL
DateOpened: DATE NULL
StudyHours: VARCHAR2(3) NULL
DateClosed: DATE NULL
ReserveCredits: CHAR(6) NOT NULL
CreditHours: VARCHAR2(3) NULL
Description: LONG NULL
DesignedFor: VARCHAR2(200) NULL
OnHand: NUMBER NOT NULL

is_comprised_of

is_associated_with

ADVERTISMENT
Project_ID: NUMBER NOT NULL
Description: VARCHAR2(2000) NOT NULL

is_funded_by

is_associated_with

has

is_funded_by

COURSE_COPY_MATERIAL
Material_ID: NUMBER NOT NULL (FK)
Comp_ID: VARCHAR2(10) NOT NULL (FK)
Course_ID: VARCHAR2(6) NOT NULL

JOB_AID_COPY_MATERIAL
Material_ID: NUMBER NOT NULL (FK)
Comp_ID: VARCHAR2(10) NOT NULL (FK)
JobAid_ID: VARCHAR2(10) NOT NULL

STUDENT
StudSSN_ID: VARCHAR2(9) NOT NULL
MCTFSSN: CHAR(9) NULL (FK)
LastName: VARCHAR2(20) NOT NULL
FirstName: VARCHAR2(10) NOT NULL
MidInit: CHAR(1) NULL
MOS: CHAR(4) NULL
PlatoonCode: VARCHAR2(4) NULL
Addr1: VARCHAR2(46) NULL
Addr2: VARCHAR2(46) NULL
City: VARCHAR2(34) NULL
ZipCode: VARCHAR2(9) NULL

has_many

has_many

1

FINANCIAL
LineItem: NUMBER NOT NULL
Year: VARCHAR2(4) NOT NULL
Program_ID: VARCHAR2(6) NULL (FK)
Crs_ID: VARCHAR2(6) NULL (FK)
BudgetAmount: NUMBER NOT NULL
Description: VARCHAR2(2000) NOT NULL
Department: VARCHAR2(40) NOT NULL

PROGRAM_COPY_MATERIAL
Material_ID: NUMBER NOT NULL (FK)
Comp_ID: VARCHAR2(10) NOT NULL (FK)
Program_ID: VARCHAR2(6) NOT NULL

LEGEND:

Silver Tables: Main Entities (COURSE, PROGRAM, STUDENT, COURSE__COMPONENTS)
White Tables: Dependent Entities and Independent Entities other than Main Entities

Blue Relationships: Involve Main Entities
Fuchsia Relationships: Involve Dependent Entities
Green Relationships: Involve Independent Entities other than Main Entities

Blue Attributes: Key/Foreign Key from Main Entity
Fuchsia Attributes: Key/Foreign Key from Dependent Entity
Green Attributes: Key/Foreign Key from Independent Entity other than Main Entity

IS_EQUIPMENT_INVENTORY
Serial_No: NUMBER NOT NULL
Nomenclature: VARCHAR2(50) NOT NULL
Description: VARCHAR2(200) NOT NULL
Location: VARCHAR2(30) NOT NULL
Custodian: VARCHAR2(30) NOT NULL

APPENDIX I. MCI ENTERPRISE LEVEL MATRICES

EXHIBIT 3: DATA SUBJECTS VS. FUNCTIONS (CRUD MATRIX)

CRUD Matrix
Data Subject vs Function

DATA SUBJECT \ FUNCTION	Planning	Budgeting	Training	Advertising	Analysis of Effectiveness	Program Design	Program Writing	Program Staffing	Program Revising	Course Design	Course Writing	Course Staffing	Course Revising	Editing	Layout & Graphics	Repro Servicing	Student Activity Transactions	Grading	Customer Servicing	Registrar Servicing	Manag Netwo
Advertisement *Information*				C R U D	R	R		R		R		R							R		
Components *Information*		R			R	R	C R U D	R	R U		C R U D	R	R U	R	R U	R	R		R		
Copy Material *Information*			R		R	R	C R U A	R		R	C R U A	R	R U	R U	C R U A	R			R		
Course *Information*		R		R	R	C R U A	R U	R	R U	C R U A	R U	R	R U	R	R	R	R	R	R	R	
Course Copy Material *Information*					R	R	C R U D	R	R U	R	C R U A	R	R U	R U	R U	R					
Course Developers *Information*					R	C R U D	R	R	R	C R U D	R	R	R	R	R				C R U D		
Customer *Information*																	R				
Events *Information*	C R U D																				
Exam *Information*					R	R	C R U A	R	R U	R	C R U A	R	R U	R U	R	R	R	R	R	R	
Financial *Information*		C R U D	R	R		R				R						R				R	R
Inventory *Information*					R												R		R		
IS Equipment Inventory *Information*		R																			C R
Issue Complaint *Information*																			C R U D		
Job Aid *Information*		R	R	R						R	C R U D	R	R U	R	R	R	R		R		
Job Aid Copy Material *Information*										R	C R U D	R	R U	R U	R U	R					
MCI Personnel *Information*	R		R																		
MCTFS Personnel *Information*																	R	R	R	R	
Order Information		R															R U		C R U D		
Program *Information*		R	R	R	R	C R U A	R U	R	R U					R	R	R	R	R	R	R	
Program Copy Material *Information*					R	R	C R U D	R	R U					R U	R U	R					
Program Developers *Information*					R	C R U D	R	R	R					R	R						
Purchase *Information*		R		R				R				R				C R U D	R		R		
SSD Personnel *Information*																					
Student *Information*					R												C R U A	R U	C R U A	R U	
Training *Information*			C R U D			R	R		R	R	R		R	R	R						
Warehouse *Information*		R																			

C - Create	R - Read	D - Delete
U - Update		A - Archive

CRUD Matrix
Data Subject vs Function

Course Staffing	Course Revising	Editing	Layout & Graphics	Repro Servicing	Student Activity Transactions	Grading	Customer Servicing	Registrar Servicing	Manage Networks	Programming	Database Administration	Ordering	Receiving	Storing	Inventory	Packaging	Delivering	Disposing	Unit Reconciling	Exam Proctoring	Parade	Personnel Administration
R							R				R											
R	R U	R	R U	R	R		R			R	R	R	R	R	R	R	R	R				
R	R U	R U	C R U A	R						R												
R	R U	R	R	R	R	R	R	R		R	R			R	R	R	R	R	R	R		
R	R U	R U	R U	R						R	R											
R	R	R	R								R											R
					R		C R U D			R	R						R		R			
																					R U	
R	R U	R U	R	R	R	R	R	R		R	R				R	R	R	R		R		
				R				R	R	R		R	R		R	R	R					
					R		R		C R U D	R	R	R U	R U	R U	C R U D	R U	R U	R U	R			
							C R U D			R	R											
R	R U	R	R	R	R		R			R	R	R	R		R U	R	R	R	R			
R	R U	R U	R U	R						R												
																						C R U D
					R	R	R	R		C R U D	R											
					R U		C R U D			R	R				R	R	R		R			
		R	R	R		R	R	R		R	R			R		R	R		R			
		R U	R U	R						R	R							R				
		R	R								R											R
R				C R U D	R		R			R	R	C R U D	R U	R	R U							
										R	R											C R U D
					C R U A	R U	C R U A	R U		R	R						R		R	R		
	R	R	R							R	R											R
										R				R U	C R U D			R				

C - Create	R - Read	D - Delete
U - Update		A - Archive

APPENDIX P. SSD AND MIS LEVEL DATA MODEL

MARINE CORPS INSTITUTE
STUDENT SERVICES DIVISION (SSD)
IDEF1X DIAGRAM

Current Version
VERSION 97156 (5 June 1997)
FILE NAME MCIV97156.ER1
mcl2/c:/mcl/data model

DEVELOPED BY MAJ. A.T. SLAUGHTER

LEGEND:

Yellow Tables: Domain Tables
Silver Tables: Main Entities (COURSE, PROGRAM, STUDENT, COURSE__COMPONENTS)
White Tables: Dependent Entities and Independent Entities other than Main Entities

Blue Relationships: Involve Main Entities
Red Relationships: Involve Domain Tables
Fuchsia Relationships: Involve Dependent Entities
Green Relationships: Involve Independent Entities other than Main Entities

Blue Attributes: Key/Foreign Key from Main Entity
Red Attributes: Key/Foreign Key from Domain Tables
Fuchsia Attributes: Key/Foreign Key from Dependent Entity
Green Attributes: Key/Foreign Key from Independent Entity other than Main Entity

ANS_CODE
AnsCode_ID: VARCHAR2(1) NOT NULL
AnsCodeDesc: VARCHAR2(30) NOT NULL

CAT_CODE
CatCode_ID: VARCHAR2(2) NOT NULL
CatCodeDesc: VARCHAR2(20) NOT NULL

CRS_PREREQ
JRCRS_ID: VARCHAR2(6) NOT NULL (FK)
SRCRS_ID: VARCHAR2(6) NOT NULL (FK)

PRG_PREREQ
SrProg_ID: VARCHAR2(6) NOT NULL (FK)
JrProg_ID: VARCHAR2(6) NOT NULL (FK)

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

STAT_CODE
StatCode_ID: VARCHAR2(2) NOT NULL
StatCodeDesc: VARCHAR2(60) NOT NULL

has_a
Defines_Grading_Method
defines_status_of
has_a_prereq_of
is_a_prereq_for
prereq_for
prereq_of

MULT_CHOICE
QuestNo: NUMBER NOT NULL
Exam_ID: VARCHAR2(2) NOT NULL (FK)
Crs_ID: VARCHAR2(6) NOT NULL (FK)
AnsCode_ID: VARCHAR2(1) NOT NULL (FK)
ExamRef_1: VARCHAR2(10) NOT NULL
ExamRef_2: VARCHAR2(10) NOT NULL

CRS
Crs_ID: VARCHAR2(6) NOT NULL
RemMeth_ID: VARCHAR2(1) NOT NULL (FK)
CrsNo: VARCHAR2(4) NOT NULL (IE1)
StatCode_ID: VARCHAR2(2) NOT NULL (FK)
CrsTitle: VARCHAR2(60) NOT NULL
CrsAbbr: VARCHAR2(20) NOT NULL
CrsDateOpen: DATE NULL
CrsStudyHours: NUMBER NULL
CrsDateClsd: DATE NULL
CrsResCred: VARCHAR2(3) NULL
EdLev: VARCHAR2(4) NULL (FK)
CrsCreditHrs: VARCHAR2(3) NULL
CrsDesc: LONG NULL
CrsDesignedFor: VARCHAR2(200) NULL
CrsOnHandQty: NUMBER NOT NULL
C_GradePrs: CHAR(2) NULL
C_PME: CHAR(1) NOT NULL
C_Accr: CHAR(1) NOT NULL

ED_LEV
EdLev: VARCHAR2(4) NOT NULL
EdLevDesc: VARCHAR2(20) NOT NULL

is_assoc_with
is_part_of
is_associated_with
is_made_up_of

CRS_PRG_X
Crs_ID: VARCHAR2(6) NOT NULL (FK)
Prg_ID: VARCHAR2(6) NOT NULL (FK)

PROGRAM
Prg_ID: VARCHAR2(6) NOT NULL
RemMeth_ID: VARCHAR2(1) NOT NULL (FK)
StatCode_ID: VARCHAR2(2) NOT NULL (FK)
PrgTitle: VARCHAR2(60) NOT NULL
ProgAbbr: VARCHAR2(20) NOT NULL
PrgDateOpened: DATE NULL
PrgDateClosed: DATE NULL
PrgStudyHrs: VARCHAR2(3) NULL
EdLev: VARCHAR2(4) NULL (FK)
PrgCreditHrs: VARCHAR2(3) NULL
SchoolCode: CHAR(3) NOT NULL
PrgOnHandQty: NUMBER NOT NULL
P_GradeEIUSMC: CHAR(2) NULL
P_GradeEIOther: CHAR(2) NULL
P_Accr: CHAR(1) NULL
Purpose: VARCHAR2(2000) NULL
Prerequisites: VARCHAR2(500) NULL
AreasOfStudy: VARCHAR2(1000) NULL
CourseAdministration: VARCHAR2(2000) NULL
CourseDeveloper: VARCHAR2(50) NULL

may_consist_of

PRG_COMP_X
Prg_ID: VARCHAR2(6) NOT NULL (FK)
Comp_ID: VARCHAR2(10) NOT NULL (FK)
PrgCompQty: NUMBER NOT NULL

JOB_AID
JobAid_ID: VARCHAR2(10) NOT NULL
JAName: VARCHAR2(30) NOT NULL
JADesc: VARCHAR2(600) NOT NULL
JAOnHandQty: NUMBER NOT NULL

is_comprised_of

JOB_AID_COMP_X
Comp_ID: VARCHAR2(10) NOT NULL (FK)
JobAid_ID: VARCHAR2(10) NOT NULL (FK)
JACompQty: NUMBER NOT NULL

EXAM
Exam_ID: VARCHAR2(2) NOT NULL
Crs_ID: VARCHAR2(6) NOT NULL (FK)
CatCode_ID: VARCHAR2(2) NOT NULL (FK)
ExmDateOpen: DATE NULL
ExmDateClose: DATE NULL
ExmPassScore: NUMBER NOT NULL
ExNoQuest: NUMBER NOT NULL

Eval_By

REM_METH
RemMeth_ID: VARCHAR2(1) NOT NULL
RemMethDesc: VARCHAR2(30) NOT NULL

Belongs_to
is_comprised_of

EXAM_COMP_X
Comp_ID: VARCHAR2(10) NOT NULL (FK)
Exam_ID: VARCHAR2(2) NOT NULL (FK)
Crs_ID: VARCHAR2(6) NOT NULL (FK)
ExmCompQty: NUMBER NOT NULL

is_ordered_as
comprises_a

ErrCntr_ID: NUMBER NOT NULL
ErrDate: DATE NOT NULL
ScanCrs_ID: VARCHAR2(6) NULL (FK)
ScanExam_ID: VARCHAR2(2) NULL (FK)
ErrCode_ID: VARCHAR2(1) NOT NULL (FK)
ScanSSN_ID: VARCHAR2(9) NULL (FK)
OpscanNumber: VARCHAR2(4) NOT NULL

is_contained_on
is_comprised_of

CRS_COMP_X
Crs_ID: VARCHAR2(6) NOT NULL (FK)
Comp_ID: VARCHAR2(10) NOT NULL (FK)
CC_Qty: NUMBER NOT NULL

is_ordered_as

have_associated

ERR_CODES
ErrCode_ID: VARCHAR2(1) NOT NULL
ErrCodeDesc: VARCHAR2(55) NOT NULL

ERR_ANS_STRING
ErrCntr_ID: NUMBER NOT NULL (FK)
ErrDate: DATE NOT NULL (FK)
ExmAnsStrng: VARCHAR2(160) NOT NULL

contains
comprises_a
may_be_included_in
is_ordered_as

STUO_CRS_EXAM_X
StudSSN_ID: VARCHAR2(9) NOT NULL (FK)
Exam_ID: VARCHAR2(2) NOT NULL (FK)
Crs_ID: VARCHAR2(6) NOT NULL (FK)
ExmDateRecv: DATE NULL
ExmScore: NUMBER NULL

have_associated

STUO_ANS
StudSSN_ID: VARCHAR2(9) NOT NULL (FK)
Exam_ID: VARCHAR2(2) NOT NULL (FK)
Crs_ID: VARCHAR2(6) NOT NULL (FK)
ExmAnsStrng: VARCHAR2(160) NOT NULL

COMP
Comp_ID: VARCHAR2(10) NOT NULL
CompName: VARCHAR2(65) NOT NULL
CompDesc: VARCHAR2(65) NOT NULL
CompOnHandQty: NUMBER NOT NULL

TRNG_NCO_MATERIAL
Item_ID: VARCHAR2(6) NOT NULL
ItemName: VARCHAR2(65) NOT NULL
ItemDescription: VARCHAR2(200) NULL
QtyOnHand: NUMBER NOT NULL

CUST
CustSSn_ID: VARCHAR2(9) NOT NULL
State: CHAR(2) NULL (FK)
CustLastName: VARCHAR2(20) NOT NULL
CustFirstName: VARCHAR2(10) NOT NULL
CustMidInit: CHAR(1) NULL
CustDSN: VARCHAR2(11) NULL
CustCommNo: VARCHAR2(14) NULL
CustAddr1: VARCHAR2(46) NOT NULL
CustAddr2: VARCHAR2(46) NULL
CustCity: VARCHAR2(34) NOT NULL
CustZip: VARCHAR2(9) NOT NULL

is_issued_a
is_ordered_as

STUDENT
StudSSN_ID: VARCHAR2(9) NOT NULL
Rank: VARCHAR2(6) NULL (FK)
Grade: CHAR(2) NULL (FK)
ServComp: CHAR(2) NOT NULL (FK)
StudLastName: VARCHAR2(20) NOT NULL
StudFirstName: VARCHAR2(10) NOT NULL
StudMI: VARCHAR2(1) NULL

Defines_Service_Affiliation_of

SVC_COMP
ServComp: CHAR(2) NOT NULL
ServCompDesc: VARCHAR2(40) NOT NULL

defines_grade_of

GRADE
Grade: CHAR(2) NOT NULL
GradeDesc: VARCHAR2(20) NOT NULL

STUO_CRS_X
Crs_ID: VARCHAR2(6) NOT NULL (FK)
StudSSN_ID: VARCHAR2(9) NOT NULL (FK)
CrsEnrolDate: DATE NOT NULL
CrsCCD: DATE NOT NULL
CrsFinalGrade: VARCHAR2(3) NULL
CrCertDate: DATE NULL
SC_PME: CHAR(1) NOT NULL
SC_Comp: CHAR(1) NOT NULL
Disenrolled: CHAR(1) NOT NULL

TRNG_NCO_LINE_ITEM
InvNo_ID: NUMBER NOT NULL (FK)
Item_ID: VARCHAR2(6) NOT NULL (FK)
ItemQty: NUMBER NOT NULL

JOB_AID_LINE_ITM
InvNo_ID: NUMBER NOT NULL (FK)
JobAid_ID: VARCHAR2(10) NOT NULL (FK)
JAQty: NUMBER NOT NULL

CRS_LINE_ITM
InvNo_ID: NUMBER NOT NULL (FK)
Crs_ID: VARCHAR2(6) NOT NULL (FK)
CrsQty: NUMBER NOT NULL

contains
is_associated_with

MCTF_PERS
MCTFSSN: CHAR(9) NOT NULL
MCTFLastName: VARCHAR2(20) NOT NULL
MCTFFirstInitial: VARCHAR2(1) NOT NULL
MCTFMidInit: VARCHAR2(1) NULL
MCTFMOS: CHAR(4) NOT NULL
MCTFBilletMOS: CHAR(4) NULL
MCTFPlatCode: CHAR(4) NOT NULL
MCTF_RUCMCC: VARCHAR2(8) NOT NULL (FK)
MCTFComp: CHAR(2) NOT NULL
MCTFRank: VARCHAR2(6) NOT NULL
MCTFGrade: CHAR(2) NOT NULL
MCTF_ECC: DATE NULL
MCTFSelToGrade: CHAR(2) NULL
MCTFAddr1: VARCHAR2(46) NULL
MCTFAddr2: VARCHAR2(46) NULL
MCTFCity: VARCHAR2(34) NULL
MCTFZip: VARCHAR2(9) NULL
MCTFState: VARCHAR2(2) NULL

is_enrolled_in
may_have_an

NON_MAR_ADDR
StudSSN_ID: VARCHAR2(9) NOT NULL (FK)
State: CHAR(2) NULL (FK)
NMSAddr1: VARCHAR2(46) NOT NULL
NMSAddr2: VARCHAR2(46) NULL
NMSCity: VARCHAR2(36) NULL
NMSZip: VARCHAR2(9) NOT NULL

STUD_CRS_TRAN_X
Crs_ID: VARCHAR2(6) NOT NULL (FK)
StudSSN_ID: VARCHAR2(9) NOT NULL (FK)
Tran_ID: CHAR(1) NOT NULL (FK)
SC_TransDate_ID: DATE NOT NULL

has_an_associated

STUD_PRG_X
Prg_ID: VARCHAR2(6) NOT NULL (FK)
StudSSN_ID: VARCHAR2(9) NOT NULL (FK)
PrgEnrolDate: DATE NOT NULL
PrgCCD: DATE NULL
PrgDiplIssueDate: DATE NULL
SP_Comp: CHAR(1) NOT NULL
Disenrolled: CHAR(1) NOT NULL

has_an_associated

INVOICE
InvNo_ID: NUMBER NOT NULL
CustSSn_ID: CHAR(9) NOT NULL (FK)
ClerkSSN_ID: CHAR(9) NULL (FK)
InvDate: DATE NOT NULL

contains

COMP_LINE_ITM
InvNo_IO: NUMBER NOT NULL (FK)
Comp_IO: VARCHAR2(10) NOT NULL (FK)
CompQty: NUMBER NOT NULL

PRG_LINE_ITM
InvNo_IO: NUMBER NOT NULL (FK)
Prg_IO: VARCHAR2(6) NOT NULL (FK)
PrgQty: NUMBER NOT NULL

has_assoc
defines_rank_of
associated_with
opens_an
contains

RUC
MCTF_RUCMCC: VARCHAR2(8) NOT NULL
UnitName: VARCHAR2(45) NOT NULL
RUCAddr1: VARCHAR2(46) NOT NULL
RUCAddr2: VARCHAR2(45) NULL
RUCCityState: VARCHAR2(36) NOT NULL
RUCZip: VARCHAR2(9) NOT NULL

RANK
Rank: VARCHAR2(6) NOT NULL
RankDesc: VARCHAR2(45) NOT NULL

defines_state_of

STATE
State: CHAR(2) NOT NULL
StateDesc: VARCHAR2(20) NOT NULL

TRANS_CODE
Tran_IO: CHAR(1) NOT NULL
TransDesc: VARCHAR2(200) NOT NULL

associated_with

STUD_PRG_TRAN_X
Tran_IO: CHAR(1) NOT NULL (FK)
Prg_IO: VARCHAR2(6) NOT NULL (FK)
StudSSN_ID: VARCHAR2(9) NOT NULL (FK)
SP_TransDate_IO: DATE NOT NULL

defines_State_of

SSO_CLERK
ClerkSSN_IO: CHAR(9) NOT NULL
ClerkLastName: VARCHAR2(20) NOT NULL
ClerkFirstName: CHAR(1) NOT NULL
ClerkMidInit: CHAR(1) NOT NULL